

MacTech

AMERICAN BOOK CENTER

F24.50

INSIDE:
C++
EXCEPTIONS

Inside InputSprocket

by Brent Schorsch

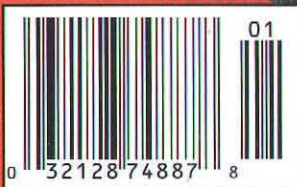
APPLE
GAMES
SPROCKETS

Also Inside:
PowerPlant LString
by John Daub

Creating FileMaker Pro Plugins
by David McKee

OpenGL on the Macintosh
by Ed Angel

\$8.95 US
\$9.95 Canada
ISSN 1067-8360
Printed in U.S.A.



choices

*The industry's only provider of ADB, USB
and software-based license management.*



SentinelEve3-USB
*All the technology
and reliability of
the SentinelEve3
in a USB model*



SentinelEve3
*The leading ADB-based
protection solution for
Macintosh developers*

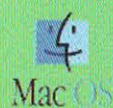
SentinelLM
*comprehensive license
management for
standalone or networks*



It's all about choices and at Rainbow Technologies we give Macintosh software developers more. As the industry leader, Rainbow offers the most secure, reliable and technologically advanced solutions to piracy and increased sales.

SentinelEve3 — ADB-based protection
SentinelEve3-USB — ready for the new iMac
SentinelLM-Mac — a software-based license manager for standalone and network applications including Java

To learn more about the choices for secure licensing, call now or visit our website www.rainbow.com/macintosh. You'll see that for Macintosh developer solutions, there's only one choice — Rainbow.



RAINBOW
TECHNOLOGIES

Changing the way the world secures business.

50 Technology Drive, Irvine, CA 92618
800.705.5552 tel. 949.450.7300 fax. 949.450.7450 www.rainbow.com/macintosh

Offices and distributors located worldwide.

1998 SentinelEve3, SentinelEve3-USB and SentinelLM are trademarks of Rainbow Technologies, Inc. All other products are trademarks of their respective owners.

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."

– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."

– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."

– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"

– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."

– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."

– Joe Zobkiw, author of A Fragment of Your Imagination

"Resorcerer will pay for itself many times over in saved time and effort."

– MacUser review

"The template that disassembles PICT's is awesome!"

– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensable in its own creation!"

– Doug McKenna, author of Resorcerer



Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: \$256 (decimal)

Website price: \$128 - \$256

(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMAESTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New
in
2.0:

- Very fast, HFS browser for viewing file tree of all volumes
- Extensibility for new Resorcerer Apprentices (CFM plug-ins)
- New AppleScript Dictionary ('aete') Apprentice Editor
- MacOS 8 Appearance Manager-savvy Control Editor
- PowerPlant text traits and menu command support
- Complete AIFF sound file disassembly template
- Big-, little-, and even mixed-endian template parsing
- Auto-backup during file saves; folder attribute editing
- Ships with PowerPC native, fat, and 68K versions

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TEMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail**?

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

DEPARTMENTS**Orders, Circulation, & Customer Service****Press Releases****Ad Sales****Editorial****Programmer's Challenge****Online Support****Accounting****Marketing****General****Web Site (articles, info, URLs and more...)****E-Mail/URL**

cust_service@mactech.com

press_releases@mactech.com

ad_sales@mactech.com

editorial@mactech.com

prog_challenge@mactech.com

online@mactech.com

accounting@mactech.com

marketing@mactech.com

info@mactech.com

http://www.mactech.com

MacTech Magazine

MacTech Magazine is grateful to the following individuals who contribute on a regular basis. We encourage others to share the technology.

We are dedicated to the distribution of useful programming information without regard to Apple's developer status.

*For information on submitting articles, ask us for our **writer's kit** which includes the terms and conditions upon which we publish articles.*

Editorial Board of Advisors

Jordan J. Dea-Mattson,
Dave Mark, Jim Straus,
and Jon Wiederspan

Editorial Staff

Publisher • Neil Ticktin
Editor Emeritus • Eric Gundrum
Editor • Nick "nick.c" DeMello
Managing Editor • Jessica Courtney
Online Editor • Jeff Clites
Web Editors • Kevin Avila, Scott Anchin

Contributing Editors

- Jim Black, Apple Computer Inc.
- Michael Brian Bentley
- Tantek Çelik, Microsoft Corporation
- Richard Clark
- Marshall Clow, Adobe Systems
- Carl de Cordova
- Andrew S. Downs
- Jim Gochee, Connectix
- Steve Kiene, Mindvision
- Peter N Lewis
- Rich Morin
- Michael Rutman
- Rich Siegel, Bare Bones Software, Inc.
- Steve Sisak, Codewell Corporation

Regular Columnists

Getting Started • Dan Parks Sydow
Programmer's Challenge • Bob Boonstra
From the Factory Floor • Dave Mark, Metrowerks
Tips & Tidbits • Jeff Clites

XPLAIN CORPORATION**Chief Executive Officer** • Neil Ticktin**President** • Andrea J. Sniderman**Controller** • Michael Friedman**Production Manager** • Jessica Courtney**Production** • Lorin Welander II**Advertising Director** • Dale Hansman**Marketing Manager** • Nick DeMello

Events Managers • John Churchill
Susan M. Worley

Network Administrator • Chris Barrus**Accounting** • Jan Webber, Marcie Moriarty**Customer Relations** • Molly Covin

Lee Ann Pham

Susan Pomrantz

Board of Advisors • Steven Geller, Blake Park,
and Alan Carsrud



This publication is
printed on paper with
recycled content.

All contents are Copyright 1984-1999 by Xplain Corporation. All rights reserved. MacTech, Developer Depot, and Sprocket are registered trademarks of Xplain Corporation. Depot, The Depot, Depot Store, Video Depot, MacDev-1, THINK Reference, NetProfessional, NetProLive, JavaTech, WebTech, BeTech, and the MacTutorMan are trademarks of Xplain Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders. Xplain Corporation does not assume any liability for errors or omissions by any of the advertisers, in advertising content, editorial, or other content found herein. Opinions or expressions stated herein are not necessarily those of the publisher and therefore, publisher assumes no liability in regards to said statements.

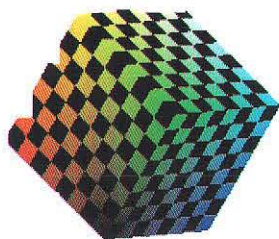
MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are \$47.00 per year. Canadian subscriptions are \$59.00 per year. All other international subscriptions are \$97.00 per year. Domestic source code disk subscriptions are \$77 per year. All international disk subscriptions are \$97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

January 1999 • Volume 15, 01

MacTech®



Feature Articles

- 12 POWER GRAPHICS**
OpenGL for Mac Users: Part 2
Advanced Capabilities: Architectural features and exploiting hardware
by Ed Angel
- 20 GAMES PROGRAMMING**
Inside InputSprocket
How to use InputSprocket to support user input in games
by Brent Schorsch

OpenGL for Mac Users..... page 12

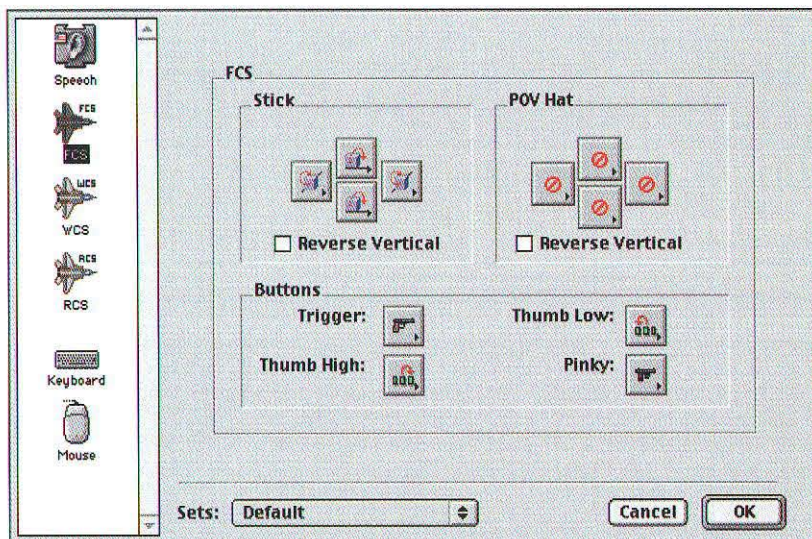
- 27 PLUGGING IN**
External Function Plug-ins for FileMaker Pro
An introduction to their nature and creation
by David McKee
- 43 POWERPLANT WORKSHOP**
The Ultra-Groovy LString Class
And introduction to the PowerPlant-way of working with strings
by John C. Daub
- 50 PROGRAMMING TECHNIQUES**
C++ Exceptions in Mac OS Code
by Steve Sisak
- 55 JAVATECH™**
Writing an OS Shell in Java
Building a Facade
by Andrew Downs

Reader Resources

- 65 NewsBits**
- 68 Classifieds**
- 70 Tips & Tidbits**
- 72 Advertiser & Product Index**

Columns

- 4 VIEWPOINT**
by Richard Gaskin
- GETTING STARTED**
- 6 Carbon: Getting Ready for Mac OS X**
by Dan Parks Sydow
- PROGRAMMER'S CHALLENGE**
- 34 Sphere Packing**
by Bob Boonstra
- FROM THE FACTORY FLOOR**
- 39 What's New at Aladdin?**
by Darryl Lovato, Brian Pfister, Peter Thomas and Dave Mark
- 66 MACTECH ONLINE**
Stocking Your Toolbox
by Jeff Clites



Inside InputSprocket..... page 20

About the cover...

This month's story focuses on Apple's InputSprocket, the technology that links user input through devices like the Ariston Ares USB Joystick and Hermes Gamepad <<http://www.ariston.com/>> to award winning games like Bungie's Myth II <<http://www.bungie.com/>> and Tomb Raider II by Aspyr Media <<http://www.aspyr.com/>>.



by Richard Gaskin

As Mac OS development tools continue to evolve, there is one important category which is often overlooked in the Mac community: rapid application development (RAD).

The availability of robust RAD tools for Windows, most notably Visual Basic, is arguably one of the primary contributors to the plethora of new applications written for the Wintel platform, and absolutely critical to the entrenchment of Wintel in corporate and academic environments where custom applications need to be cranked out regularly.

RAD tools represent a critical component of Mac evangelism as well, allowing opportunities for organizations to create custom solutions which fill market niches and keep folks using Macs. Apple has been proudly citing the number of new applications for Mac OS since the announcement of the iMac, but it seems a fair bet that this number would at least double if the company took a more active role in popularizing RAD tools for Mac OS.

In recent years, developers on other platforms have seen an increased awareness of the value of scripting. As John Ousterhout of Scriptics Corporation put it in his seminal white paper "Scripting: Higher Level Programming for the 21st Century" <<http://www.scriptics.com/people/john.ousterhout/scripting.html>>:

Scripting languages and system programming languages are complementary, and most major computing platforms since the 1960's have provided both kinds of languages. The languages are typically used together in component frameworks, where components are created with system programming languages and glued together with scripting languages. However, several recent trends, such as faster machines, better scripting languages, the increasing importance of graphical user interfaces and component architectures, and the growth of the Internet, have greatly increased the applicability of scripting languages. These trends will continue over the next decade, with more and more new applications written entirely in scripting languages and system programming languages used primarily for creating components.

Many of the most popular scripting tools on other platforms were inspired by Apple's own HyperCard, including Asymetrix ToolBook and Visual Basic. The attraction is easy to appreciate: by marrying a hierarchical GUI environment to an interpreted language which automates most memory management, development cycles can be slashed by orders of magnitude. Offering the same friendliness that is the essence of Macintosh, HyperCard introduced programming for the rest of us.

Ironically, while Apple has spent millions on since-abandoned efforts like Dylan and ScriptX, their most successful development tool, HyperCard, has never been enhanced beyond its monochrome architecture and may soon be axed <<http://www.hyperactivesw.com/SaveHC.html>>.

But in spite of Apple's lack of serious commitment to HyperCard in recent years, other vendors have successfully expanded on the easy-to-learn language and are allowing people to develop solid Mac

applications in record time for their organizations, and often for commercial distribution as well.

Many folks have the impression that any xTalk language (the collective term for HyperTalk and related dialects) is unsuitable for serious development, but these perceptions have more to do with HyperCard's implementation than any inherent problem with the language or object model. At least two other vendors have successfully enhanced xTalk in ways that are extremely potent for RAD work.

For cross-platform deployment, MetaCard <www.metacard.com> offers a very similar object model as HyperCard, but also offers multiple windows in different styles, integrated color, rich media support, and has one of the fastest interpreters in the business. Being the only xTalk which allows authoring and deployment on all major platforms (Mac OS, Windows, and UNIX), I have found it to be an excellent substitute for Java for many projects, allowing me to enjoy universal deployment at a fraction of the development time. And because scripting is essentially a shorthand glue between compiled routines in an interpreter, many operations run noticeably faster in MetaCard than equivalent Java implementations. For GUI apps, MetaCard makes Java look like "write once, crawl anywhere".

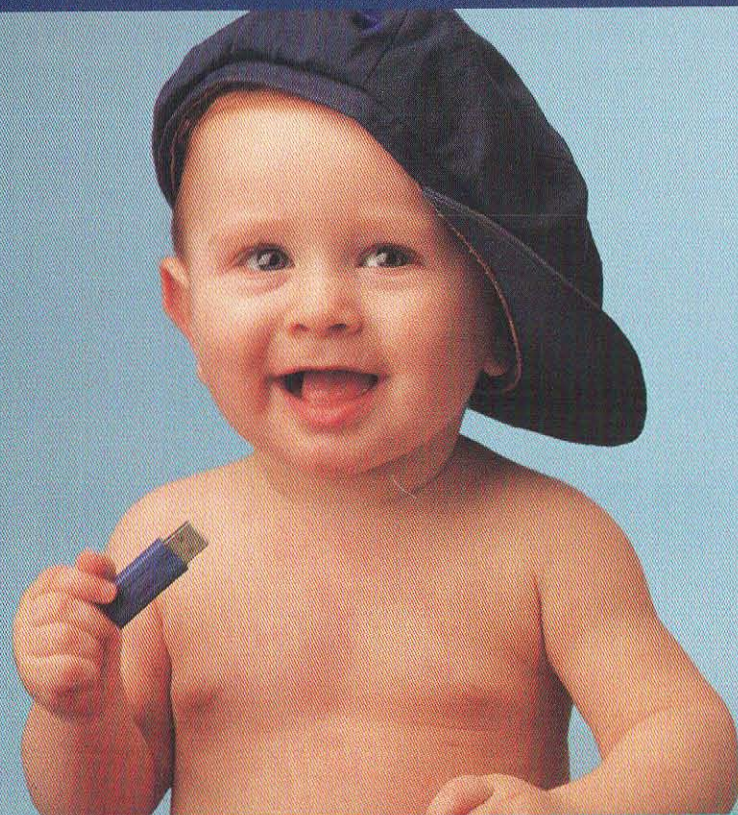
To make truly Mac-like software quickly, there is no better product than SuperCard <www.incwell.com>. While SuperCard is currently only available for Mac OS, it offers many of the advantages of MetaCard but is unencumbered by cross-platform considerations. Like a software glove wrapped around the Mac OS, SuperCard allows developers to take advantage of extremely modern Mac OS features such as the Appearance Manager, speech recognition, QTVR, and what is probably the world's first scriptable control over OS-level drag-and-drop. Try implementing all of these in C++ in an afternoon. :) The rumor mill has it that the SuperCard team is working on a suite of tools aimed at professional developers which will include an API for accessing many of the program's internal data structures and routines. This would allow developers to use SuperCard as a form of precompiled application framework, in which the standard GUI stuff is handled in scripting while computationally-intensive routines can be easily integrated from custom C code.

While there are limitations to developing with any 4GL, the advantages are compelling for many applications, especially those designed for vertical markets, schools, and smaller organizations where the greater costs of developing purely in C++ or Java would be prohibitive. In those formal languages, development time in measured in weeks and months, but in xTalk products these are often measured in days, or even hours.

And even for projects based in C++, xTalk products provide all the necessary ingredients for rapid prototyping, which can make a critical difference in early design and usability discussions. Apparently Microsoft agrees: If my sources are correct, even Visual Basic was first prototyped in SuperCard, on a Mac of course. :) MT

Richard Gaskin is the ambassador of Fourth World <<http://www.fourthworld.com>>, a Los Angeles-based consultancy specializing in scripting languages for Mac, Windows, and UNIX. You can reach him at ambassador@fourthworld.com.

MacHASP USB



MacHASP USB – So small, So easy, So powerful

- Protects software via any available USB port
- Unequaled security
- Unparalleled flexibility, ease-of-use and transparency
- Compatible with MacHASP keys and software
- Hot swappable
- Contains 100 bytes of memory
- Available in stand-alone and network versions



HASP Protects More

North America Aladdin Knowledge Systems Inc. 800 223-4277, 212 564-5678, Fax: 212 564-3377, Email: hasp.sales@aks.com
 Int'l Office Aladdin Knowledge Systems Ltd. +972 3 656-2222, Fax: +972 3 537-5796, Email: hasp.sales@aks.com
 Germany Aladdin Knowledge Systems GmbH & Co. KG +49 89 89 42 21-0, Fax: +49 89 89 42 21-40, Email: info@aladdin.de
 UK Aladdin Knowledge Systems UK Ltd. +44 1753 622266, Fax: +44 1753 622262, Email: sales@aladdin.co.uk
 Japan Aladdin Japan Co., Ltd. +81 426 60-7191, Fax: +81 426 60-7194, Email: sales@aladdin.co.jp
 France Aladdin France SA +33 1 41-37-70-30, Fax: +33 1 41-37-70-39, Email: info@aladdin.fr
 Benelux Aladdin Software Security Benelux B.V. +31 24 648-8444, Fax: +31 24 645-1981, Email: sales@aladdin.nl
 Russia Aladdin Software Security R.D. Ltd. +7 095 923-0588, Fax: +7 095 928-6781, Email: aladdin@aladdin.msk.ru

■ Australia Conlab 03 98985685 ■ China (Beijing) Feitain 010 62567389 (Hong Kong) Hastings 0755 2328741 ■ Czech Atlas 02 766085 ■ Denmark Berendsen 039 577316 ■ Egypt Zeineldin 02 3604632 ■ Finland ID-Systeme 09 8703520
 ■ Greece Unibrain 01 6756320 ■ India Solution 011 2148254 ■ Italy Partner Data 02 26147380 ■ Jordan ProgressSoft 06 5623820 ■ Korea Dae-A 02 8484481 ■ Mexico SiSoft 091 80055283 ■ Poland Systherm 061 8480273 ■ Portugal Futurmatica 01 4116269 ■ Romania
 Ro Interactive 092 353645 ■ Singapore ITR 065 5666788 ■ South Africa Adroit 011 8864704 ■ Spain PC Hardware 03 4493193 ■ Sweden Kortub 455 307 300 ■ Switzerland Opag 061 7169222 ■ Taiwan Teco 02 25559676 ■ Turkey Mikrobeta 0312 4670635

© Aladdin Knowledge Systems Ltd. 1995-1998 HASP and (B) 1998 1.0-4.1 HASP is a registered trademark of Aladdin Knowledge Systems Ltd. All other product names are trademarks of their respective owners.

Protects Your Software

Your software is your baby – and you want to look after it. You created it, you developed it, you saw it right through to the moment it was ready for market. Now protect it. 50% of business software is stolen; \$11 billion of developers' income is lost to piracy.* Is your software a statistic?

All over the world, more developers are protecting against piracy. They're protecting more products, on more platforms, with more security – and selling more as a result. And more of these developers are protecting with HASP.

To see why 25,000 developers
worldwide protect with Aladdin,
call us today!



1-800-223-4277
www.aks.com

ALADDIN

The Professional's Choice

* 1998 BSA/SPA Study

by Dan Parks Sydow

Carbon: Getting Ready for Mac OS X

Becoming familiar with the Carbon API to ready your code for Mac OS X

In the previous few months we've covered a variety of graphics-related programming topics, including bringing color and animation to your Mac programs. In future articles we'll discuss other multimedia enhancements (such as sound-playing) that you can incorporate into your own code. This month, however, we'll take a pause in the multimedia action to look at what is becoming a very important topic to all Mac programmers: Mac OS X and the Carbon application programming interface.

Apple will seed the beta version of Mac OS X to select developers next month. Some of you readers may be in the group lucky enough to be getting your hands on a copy of the new, modern operating system. But even if you won't be getting a sneak-peek at Mac OS X, there are still plenty of things you can do to ensure that your own program — whether trivial or all-powerful — is ready for the prime-time release of Mac OS X later this year.

ABOUT MAC OS X

By now you've certainly read numerous magazine columns, online reports, and even newspaper articles, devoted to Mac OS X. So we can be succinct here in our definition of just what Mac OS X is. Mac OS X is the version of the Macintosh operating system that's to bring about the important new features and enhancements that all developers and

many users have clamored for. Applications running in Mac OS X will have a number of advantages over applications running in previous versions of the Mac OS X. In short, these advantages are:

Increased stability A protected address space means memory protection. When a program "goes bad" it may crash, but it won't take down other applications or crash the system.

Increased responsiveness Preemptive multitasking means that the processor is better able to service all running applications, resulting in each application being more responsive.

Dynamic resource allocation Applications use system resources (such as memory) as needed — they don't need to specify and adhere to predetermined values.

Okay, Mac OS X sounds pretty cool — users will really experience faster applications and fewer crashes. But what do these enhancements mean for us programmers? The answer to that question is the basis of this article, so read on!

ABOUT CARBON

API stands for application programming interface, and it's the means of helping programmers develop applications that make use of, or interface with, the code that comprises the operating system. Each operating system has its own API, so to make it quickly evident which API a programmer is referring to, each API has a name. As you know from reading *Getting Started*, the API Macintosh programmers use is referred to as the Macintosh Toolbox (or the Mac Toolbox, or simply the Toolbox). The Macintosh Toolbox, like any API, is nothing more than a set (albeit a very large set) of routines. When you learn how to make use of these routines, you know how to write programs.

Mac OS X will introduce a host of changes to the Macintosh operating system, so a new means of allowing programmers to interface with the OS is needed. Carbon is that means. Carbon, like the Toolbox, is the set of programming interfaces a programmer uses to develop Macintosh applications. Specifically, Carbon is the set of routines programmers use to develop Mac OS X applications that can also run on Mac OS 8. When a programmer writes a program using Carbon, that program takes advantage of the features new to Mac OS X, such as preemptive multitasking, memory protection, and dynamic resource allocation.

What happens when you run a "carbonized" application on a Mac hosting Mac OS 8 rather than on a Mac hosting Mac OS X?

It will run just fine. It won't, however, be more responsive or more stable than any other Mac OS 8 application. Mac OS 8 will simply consider it another Mac OS 8 application. Copy that same application to a Mac hosting Mac OS X, though, and the carbonized program will exhibit the traits expected of a Mac OS X application. To summarize:

Macintosh Toolbox The API for developing applications that run on a Macintosh computer hosting any version of the Mac OS prior to Mac OS X (such as Mac OS 8.5, Mac OS 7.6, and so forth).

Carbon The API for developing applications that can run on a Macintosh computer hosting Mac OS X or any version of Mac OS 8.

Learning a new API can be a daunting task. If you're a regular reader of *Getting Started*, you are probably just beginning to feel comfortable with using the Macintosh Toolbox API. If tomorrow you were asked to develop a program that would run on a PC hosting Windows 95 or Windows 98, you'd have your work cut out for you. While some general programming principles would translate from one operating system to the other, your API knowledge would be all but useless — Windows has its own huge set of API routines that a programmer uses to develop a Windows application. This all sounds pretty ominous, but it isn't. It's just a setup to let you know how easy you, the Mac programmer, has it. Why? Because the transition from the Macintosh Toolbox API to the Carbon API is nothing like the transition from the Macintosh API to the Windows API. This is because Carbon is nothing more than a cleaned-up, enhanced, Macintosh Toolbox. If you're familiar with the Macintosh Toolbox, you're familiar with Carbon. The more you already know about the Macintosh Toolbox, the more you will know about Carbon.

When Apple bought NeXT, they obtained OpenStep — the API that NeXT programmers use to write applications that run on a computer hosting the NeXT operating system. Just as a Mac programmer makes use of the Toolbox API (and soon, the Carbon API) when developing a Macintosh application, a NeXT programmer makes use of the OpenStep API when developing a NeXT application. After acquiring NeXT, Apple set about adding to OpenStep — Apple engineers wrote new code to enhance the existing OpenStep code. To distinguish this "enhanced OpenStep" from the original OpenStep, Apple gave the new API a new name: Yellow Box. Like Carbon, the Yellow Box is an API used to develop applications that will run on Mac OS X. Unlike Carbon, the Yellow Box isn't similar, or based on, the Macintosh Toolbox (because its origin is a different operating system — the NeXT operating system).

The disadvantage to developing with the Yellow Box is readily apparent: a Mac programmer needs to learn a completely new API. The advantage to developing with the Yellow Box, though, can be impressive: applications based on the Yellow Box API are cross-platform. An application developed using the Yellow Box will, without modification, run on a Macintosh hosting Mac OS X. It will also run on a PC running Yellow Box for Windows — the Apple software that can be installed on most PCs running Windows 95 or Windows 98. When a PC user has Yellow Box for Windows on his Wintel computer, his machine

can run all his Windows applications and any Mac OS X application developed using the Yellow Box API.

Programming with the Yellow Box API is out of the scope of *Getting Started* — here we'll stick to the tried-and-true Macintosh Toolbox and its successor, Carbon. Look for a number of Yellow Box articles in other areas of upcoming issues of *MacTech Magazine*.

PREPARING FOR CARBON

Mac OS X isn't here yet, and the Carbon API isn't finalized either. But of course we won't let those minor obstacles prevent us from getting started programming for Mac OS X! Even without a final version of Carbon available, you can write Carbon-compliant code. Carbon-compliance means:

Compilation Your code successfully compiles when using the same universal interface files that will be used for creating Mac OS X applications.

Mac OS X and Mac OS 8 compatible The resulting application runs on both Mac OS 8 and Mac OS X.

Mac OS X enhanced The resulting application takes advantage of Mac OS X features such as preemptive multitasking when running on Mac OS X.

USING THE UNIVERSAL INTERFACE HEADER FILES

One step in the preparation for Carbon is to obtain and use the most recent version of the universal header files. These interface files provide your projects with the latest function prototypes for all of the Toolbox functions. As Apple alters Toolbox functions for Carbon, the changes will be reflected in the function prototypes in the universal header files. These interface files are always available to programmers through Apple's public Web site — go to Apple's Carbon Web site at <http://gemma.apple.com/macosx/carbon/> to find the link that lets you download the whole set.

When you installed CodeWarrior on your hard drive, the installer copied the universal header files from the install CD to a folder named Universal Headers in the Headers folder — the full path from inside your main CodeWarrior folder is Metrowerks CodeWarrior:MacOS Support:Headers:Universal Headers. Replace this Universal Header folder with the newly downloaded folder of the same name (you can always return to using the original set by copying the universal header files from the CodeWarrior install CD).

CodeWarrior projects typically make use of the universal header files by including one of the compiled MacHeaders libraries (they reside in the MacHeaders folder, which from your main CodeWarrior folder has a path of Metrowerks CodeWarrior:MacOS Support:MacHeaders). To get your projects to make use of the new universal header files you'll need to recompile the various MacHeader libraries. CodeWarrior comes with an AppleScript that takes care of that task for you — from the AppleScript menu of the CodeWarrior IDE choose Recompile MacHeaders. If the IDE isn't displaying the AppleScript menu, check the Use Script menu checkbox from the IDE Extras panel in the IDE's preferences window (choose Preferences from the

Edit menu). Once the AppleScript executes, the MacHeaders libraries will be updated and your existing and new projects will make use of them without any extra effort on your part.

BECOME FAMILIAR WITH FUNDAMENTAL TOOLBOX DIFFERENCES

The Mac OS X application model and the Mac OS 8 application model are essentially the same. Each type of program is centered on an event loop that calls `WaitNextEvent()` to procure events from the event queue, and each type of program uses Human Interface routines from the Toolbox to add and support interface elements such as menus and windows. Mac OS X and Mac OS 8 do differ enough from one another in that programs won't access system services in identical manners, though. Here are a few of the noteworthy differences.

Human Interface Toolbox This is the part of the Macintosh Toolbox that consists of a group of routines that implement the Mac OS human interface. Menu, window, and dialog box routines are all found here. For the most part, the Human Interface Toolbox is supported. The major exception is that the Standard File Package will not be supported. Instead, Mac OS X will rely on Navigation Services. Read more about Navigation Services in the August 1998 issue of MacTech Magazine (Vol.14, No.9) and in the Programmer's Introduction to Mac OS 8.5 article in the November 1998 MacTech Magazine issue (Vol.14, No.11).

Application Utilities This area of the Macintosh Toolbox holds the routines that extend the human interface. Apple Guide and the Drag Manager fall into this group. Mac OS X promises to support most of these utilities. One possible exception is the Speech Manager — Apple hasn't come to a final decision as to whether the Speech Manager routines will remain supported.

Graphic Services The Macintosh Toolbox functions that are thought of as graphic services routines are the ones that allow your program to display images. QuickDraw GX won't be supported in Carbon, but the more popular QuickDraw and QuickDraw 3D managers will be supported. Color QuickDraw, the topic of Getting Started two issues back, will also be supported.

Multimedia Services QuickTime is the most popular means of getting multimedia effects into a Macintosh program. QuickTime will be fully supported in Carbon.

Low-Level Operating System Services Designing and implementing an application's user-interface is enjoyable because you get visual feedback of your efforts. But the low-level OS services, such as memory management and file handling, are equally important. Carbon will support most of these services, but expect some reworking of your source code. Memory management in Mac OS X will differ from Mac OS 8, so flag (comment) any memory management code in your projects to remind you that changes may be in order. The same applies to your project's file handling code. Most file handling code will work as is, but the information necessary to define a file system specification (an `FSSpec` variable) will need to be more complete than in the past. Future Getting Started columns will tackle a couple of often-neglected low-level services: memory management and file handling.

TESTING FOR CARBON COMPATIBILITY

Apple is going through the Macintosh Toolbox and determining which of the thousands of routines should be cut, which should stay in an unaltered state, and which should stay but undergo changes. You can find out the status of any Toolbox routine by clicking on the Carbon Specification link at <http://gemma.apple.com/macosex/carbon/>. Using this method to check the Carbon-compatibility of each Toolbox routine your application calls would be a laborious process, so Apple's come up with a software tool that automates the operation. The Carbon Dater application, freely available from Apple at <http://developer.apple.com/macosex/>, helps you determine the amount of effort you'll need to expend to port a Macintosh application to Mac OS X.

THE CARBON DATER ANALYZER

The Carbon Dater analyzer program examines any PowerPC application and generates an output file that can be further examined for Carbon-compatibility. You email this output file to Apple, and Apple then analyzes its contents and returns via email a final compatibility report in HTML format. The process is automated and fast — you may have your report back within the hour!

THE ANALYZER REPORT

The output data file created by the Carbon Dater program is an interim file — it alone isn't helpful to you. Instead, you want to read Apple's Carbon-compatibility report that gets generated from the interim Carbon Dater file.

Apple's report lists a number of potential pitfalls that you should watch for in writing code that you want to run on Mac OS X. Most importantly, the report examines all the Toolbox functions your code accesses and determines which calls you need to update or replace. Every questionable Toolbox call your application makes is mentioned in the report. Specifically, the report tells whether a questionable function is supported but modified, unsupported, under evaluation (as to whether it will be supported), or doesn't exist in the latest version of the universal header files.

As shown later in this article, the report includes a pie chart that shows the percentages of Toolbox calls that fall into each category. In addition to alerting you to questionable calls, the report may provide solutions that serve as workarounds.

CARBON DATING SOME CODE

You can run any PowerPC code you want through the Carbon Dater, but until you're familiar with the Carbon Dating process, it makes sense to avoid working with a monolithic application. Instead, make your initial test one that uses a relatively trivial program. Choose your own simple application, or pick any of the examples from previous Getting Started columns. For no particular reason we'll look over the PixMapper example from last month's Getting Started. You can download that code (or any other back-issue code) from the MacTech ftp site at <ftp://ftp.mactech.com/src/>.

You'll be stuck in the heaviest traffic in San Francisco and loving every minute of it.

This year at Macworld Expo, AppleR is going to make it easier for all developers, big and small, to get a space as special as their ideas. *MacTech Magazine* presents "Developer Central," an exhibit at Macworld that showcases development for the MacintoshR in the biggest possible way. Macworld will be held in San Francisco, January 5-8, 1999, in the North Hall of the Moscone Convention Center. For information on how to participate, check out the Exhibitor Information: www.devcentral.com



Apple Developer Connection



RUNNING THE CARBON DATER

The Carbon Dater only works with PowerPC-native applications, so set your CodeWarrior project to generate a PowerPC or fat application and then build an application. From the desktop open the Carbon Dater folder and drag your application's icon onto the Carbon Dater icon. When you do that, the Dater goes to work. As shown in **Figure 1**, the Carbon Dater posts a window that provides you with a little feedback as to what the analyzer is doing.

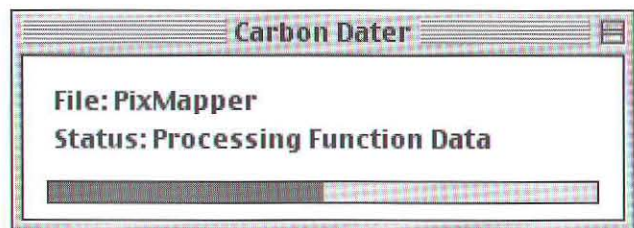


Figure 1. The Carbon Dater in action.

When the Carbon Dater is finished, you're prompted to enter your email address. This address should represent the address you want Apple's final report to be sent to, and will be embedded in the Carbon Dater output file. **Figure 2** shows this prompt.

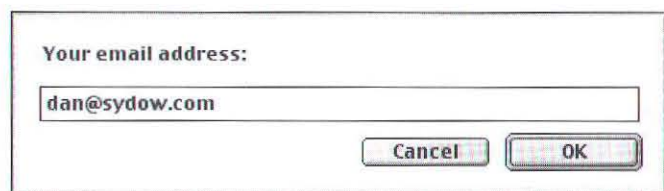


Figure 2. Telling Apple where to send the report.

After you dismiss the email dialog box, the program responds by telling you where to email the Carbon Dater output file. **Figure 3** shows that the output file will be your program's name with the extension .CCT appended to it, and that this file goes to CarbonDating@apple.com.

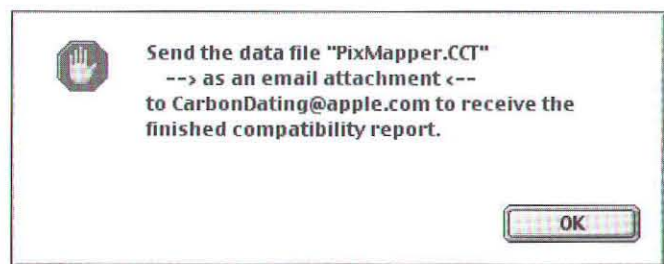


Figure 3. Getting a reminder as to what to do with the Carbon Dater output file.

With the initial analysis complete, it's time to give Apple the opportunity to provide you with the final decision on the Carbon-compatibility of your code. Launch your email application of choice, create a new mail message, and address it to CarbonDating@apple.com. Find the .CCT data file on your hard drive and add it to the message as an attachment. While it may not always be necessary, Apple recommends that you send the .CCT file as a stuffed file. If your email program supports stuffing, enable that feature to place the .CCT file in a Stuffit archive. If your email program won't do that for you, use one of Aladdin's programs, such as Stuffit or DropStuff, to stuff the .CCT file and then attach it to the email message. The message subject and body will be ignored, so at this point you're all set to send the message off to Apple.

The chief responsibility of the Carbon Dater is to extract the names of the Toolbox routines your application calls. That information goes into the .CCT data file produced by the Carbon Dater. Apple's job is to look over the contents of the .CCT file, create a final report, and email that report to the address you specified when you ran the Carbon Dater. These tasks are all automated, so after emailing the .CCT file to Apple, expect a returned report within an hour or so.

LOOKING OVER THE REPORT

Apple's report to you will be in the form of an HTML file — open it with any browser. The report begins with a summary like the one shown in **Figure 4**. This figure shows that the PixMapper program is over ninety percent Carbon-compliant. Apple has gone to great lengths to ensure that porting existing Mac OS 8 code to Mac OS X code is as painless as possible, so unless your code performs exotic, non-recommended tricks, it too should be very compliant.

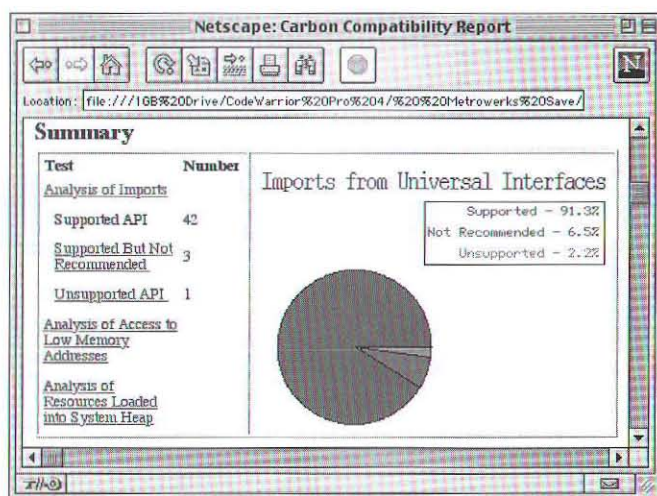


Figure 4. Looking over the final Carbon-compatibility report.

The report's summary shows that PixMapper includes calls to four questionable Toolbox functions. The next two paragraphs are what the report has to say about one of the

three routines that fall into the Supported But Not Recommended category:

Carbon will support the Dialog Manager. However, Apple encourages you to ensure that your application accesses Dialog Manager data structures only through the supplied accessor functions. The use of these accessor functions, to be provided soon, will give your application greater threading flexibility in Mac OS X. Furthermore, you are encouraged to use the functions provided for creating and disposing of Dialog Manager data structures. In Mac OS X, applications might not be allowed to create and dispose of Dialog Manager data structures except by calling Dialog Manager functions.

InitDialogs InitDialogs does nothing. There is no need to initialize the Window Manager, because the shared library gets loaded as needed.

PixMapper's call to InitDialogs() is flagged because the call won't be necessary in Mac OS X applications. The Dialog Manager won't need to be initialized because in Mac OS X its code gets loaded into memory. A Mac application can still make the call, but it will be unnecessary.

Just before the information on InitDialogs(), the report includes information regarding Carbon's support of the Dialog Manager. Don't be alarmed by these "warnings" that appear throughout the report — they may or may not apply to your code. Each modified or unsupported Toolbox routine that your program makes generates a short, general essay on the Carbon-compatibility of the manager the routine belongs to. So while PixMapper doesn't make any serious Dialog Manager mistakes, the report still includes Dialog Manager information (because of the call to the unsupported Dialog Manager routine InitDialogs()). In short, we're being reminded to stick to using the Toolbox when working with dialog boxes. We shouldn't try to circumvent Apple's way of doing things by developing our own routines to access fields of a dialog record. As a Getting Started reader you know we try to stick to doing things by-the-book, and certainly would never try tricks like that!

Now let's see some of what the report has to say about PixMapper's one call to an unsupported Toolbox function:

In Mac OS X, code that communicates directly with hardware devices must use the IOKit API. Other types of code that have relied on the Device Manager interface in the past (such as desk accessories) should be converted into applications.

OpenDeskAcc Desk accessories will not be supported in Carbon. A new mechanism will be provided for handling selections from the Apple menu.

The IOKit is new, and particular to, Mac OS X. There is no comparable Mac OS routine set. The comment says that the concept of desk accessories is being abandoned (it has been, by the way, unofficially abandoned for awhile now). The OpenDeskAcc() routine — which we've been using in our examples to open any item in the Apple menu, won't be supported in Carbon. No replacement routine is mentioned

in the report, but at least now we know what code is subject to change. We can't immediately correct the potential problem, but as shown in **Figure 5** we can go back into the PixMapper.c file and add a comment to serve as a reminder of what needs updating in the PixMapper code.

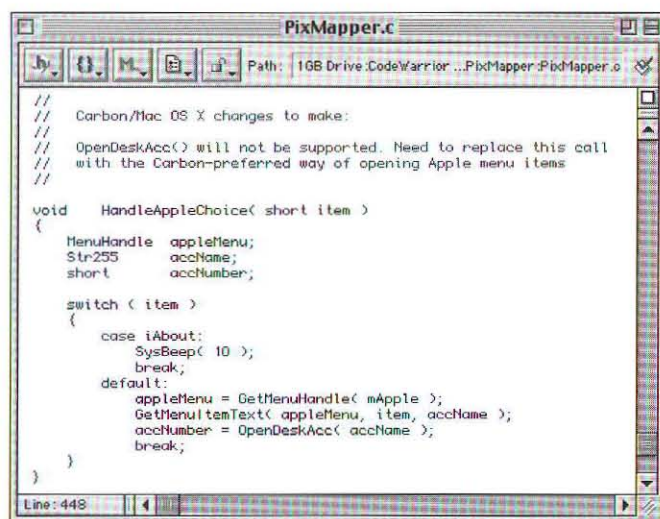


Figure 5. Flagging code for future changes.

After looking over the report you'll be able to at best make the necessary code changes, and at worst mark the potential problem areas for future alterations. In all cases you'll be closer to being Mac OS X ready.

TILL NEXT MONTH...

Apple's Mac OS X plan is clear: provide the user with a faster, more stable computing environment, and provide developers with a straightforward, short learning-curve way of getting the user there. The Mac OS X operating system is for the Macintosh user. Carbon is for you, the Macintosh programmer. Starting now, you'll want your code to be Carbon-compliant.

Now that you've seen how to Carbon date your code, do it! Don't stop at one test, Carbon date any programs to which you have the source code. It's reassuring to see returned reports that show your code to be very Carbon-compliant, but it's a good learning experience to get back a bad report too! Try to find out just what kind of code isn't acceptable for the development of a Carbon-compliant application so that you'll be truly ready for Mac OS X.

Previous Getting Started articles each included a short example program — this month's column didn't. Next month we'll return to our regular format of presenting a basic programming topic supplemented by a short, to-the-point, code example. As usual, the emphasis will be on Toolbox basics. However, now you know that what we've referred to in the past as Toolbox basics will subsequently be referred to as Carbon basics! See you next month...

MT

by Ed Angel, University of New Mexico

OpenGL for Mac Users: Part 2

Advanced capabilities: architectural features and exploiting hardware

INTRODUCTION

In Part 1, we developed the basics of OpenGL and argued that the OpenGL API provides an efficient and easy to use interface for developing three-dimensional graphics applications. However, if an API is to be used for developing serious applications, it must be able to exploit modern graphics hardware. In this article, we shall examine a few of the advanced capabilities of OpenGL.

We shall be concerned with three areas. First, we shall examine how to introduce realistic shading by defining material properties for our objects and adding light sources to the scene. Then we shall consider the mixing of geometric and digital techniques afforded by texture mapping. We shall demonstrate these capabilities through the color cube example that we developed in our first article. We will then survey some of the advanced features of OpenGL, concentrating on three areas: writing client-server programs for networked applications, the use of OpenGL buffers, and the ability to tune performance to available hardware.

Figure 1 is a more detailed view of the pipeline model that we introduced

in Part 1. Geometric objects such as polygons are defined by vertices that travel down the geometric pipeline while discrete entities such as bits and picture elements (pixels) travel down a parallel pipeline. The two pipelines converge during rasterization (or scan conversion). Consider what happens to a polygon during rasterization. First, the rasterizer must compute the interior points of the polygon from the vertices. The visibility of each point must be determined using the z or depth buffer that we discussed in Part 1. If a point is visible, then a color must be determined for it. In the simple model that we used in Part 1, a color either was assigned to an entire polygon or was interpolated across the polygon using the colors at the vertices. Here we shall consider two other possibilities that can be used either alone or together. We can assign colors based on light sources and material properties that we can assign to the polygon. Second we can use the pixels from the discrete pipeline to determine or alter the color, a process called *texture mapping*. Once a color has been determined, we can place the point in the frame buffer, place it in one of the other buffers, or use the other buffers and tables to modify this color.

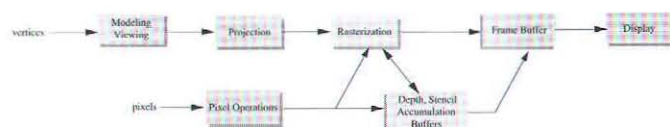


Figure 1. Pipeline Model.

In Part 1, we developed a sequence of programs that displayed a cube in various ways. These programs demonstrated the structure of most OpenGL programs. We divided our programs into three parts. The `main` function sets up the OpenGL interface with the operating system and defines the callback functions for interaction. It will not change in our examples here. The `myinit` function defines user parameters.

Ed Angel is a Professor of Computer Science and Electrical and Computer Engineering at the University of New Mexico. He is the author of *Interactive Computer Graphics: A top-down with OpenGL* (Addison-Wesley, 1997). You can find out more about him at www.cs.unm.edu/~angel or write him at angel@cs.unm.edu.

OPENBASE®

Q L D A T A B A S E E N G I N E

"OpenBase has given us tremendous flexibility in deploying cross-platform solutions. It is a rock-solid 24 x 7 performer"

Paul Summermatter LGS Systems

The Data Viewer allows you to inspect and edit database information.

MOVIE [Movie] - View

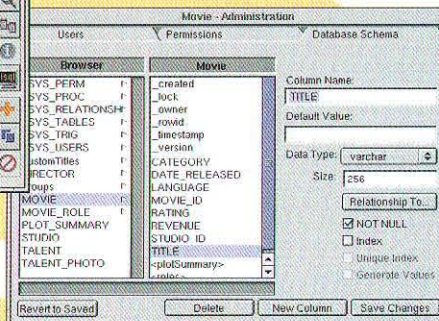
83 found

rowid	CATEGORY	TITLE	RATING	REVENUE
36	Horror	Alien	R	\$11,200,000.00
51	Comedy	Amarcord	R	\$3,218,000.00
18	Drama	Apocalypse Now	R	\$1,334,000.00
67	Drama	Bad Timing	R	\$1,000,000.00
70	Sci-Fi	Blade Runner	R	\$400,000.00
50	Drama	Blue Velvet	R	\$300,000.00
3	Drama	Cavablanca	G	\$11,200,000.00
29	Detective	Chinatown	R	\$500,000.00

SELECT rowid, CATEGORY, TITLE, RATING, REVENUE, DATE_RELEASED, MOVIE_ID, LANGUAGE, STUDIO_ID FROM MOVIE ORDER BY MOVIE_TITLE

Add Record
Remove
Execute SQL

OpenBase Manager database control panel provides complete remote control over your network databases.



The Schema Tool enables real-time database schema editing. Comprehensive GUI-based tools simplify management of user accounts.

Outrageously Fast. Surprisingly Affordable.

OpenBase is the easiest way to build and deploy powerful database applications on Mac OS X Server, Solaris and Windows NT. Order your copy today!

603-547-8404 • <http://www.openbase.com>

OPENBASE FEATURES

- 100% Java JDBC Driver
- EOF 3.0 Adapter
- Multi-Threaded Server
- Row Level Locking
- Database Management Tools
- Data Integrity Enforcement
- Cascading Deletes
- Data Clustering
- 100 MB Searchable Blobs
- Database Replication

SERVER PLATFORMS

- Mac OS X
- Solaris
- Windows NT



OPENBASE®
INTERNATIONAL

The display callback typically contains the graphical objects. Our examples here will modify these two functions.

LIGHTS AND MATERIALS

In simple graphics applications, we assign colors to lines and polygons that are used to color or shade the entire object. In the real world, objects do not appear in constant colors. Rather colors change gradually over surfaces due to the interplay between the light illuminating the surface and the absorption and scattering properties of the surface. In addition, if the material is shiny such as a metallic surface, the location of the viewer will affect what shade she sees. Although physically-based models of these phenomena can be complex, there are simple approximate models that work well in most graphical applications.

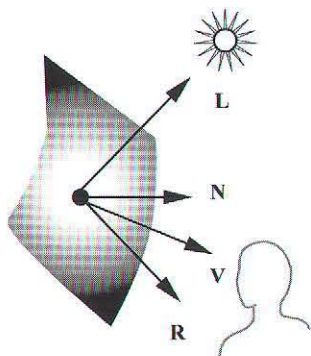


Figure 2. The Phong Shading Model.

OpenGL uses the Phong shading model, which is based on the four vectors shown in Figure 2. Light is assumed to arrive from either a point source or a source located infinitely far from the surface. At a point on the surface the vector L is the direction from the point to the source. The orientation of the surface is determined by the normal vector N . Finally, the model uses the angle of a perfect reflector, R , and the angle between R and the vector to the viewer V . The Phong model contains diffuse, specular, and ambient terms. Diffuse light is scattered equally in all directions. Specular light reflects in a range of angles close to the angle of a perfect reflection, while ambient light models the contribution from a variety of sources and reflections too complex to calculate individually. The Phong model can be computed at any point where we have the required vectors and the local absorption coefficients. For polygons, OpenGL applies the model at the vertices and computes vertex colors. To color (or shade) a vertex, we need the normal at the vertex, a set of material properties, and the light sources that illuminate that vertex. With this information, OpenGL can compute a color for the entire polygon. For a flat polygon, we can simply assign a normal to the first vertex and let OpenGL use the computed vertex color for the entire face, a technique called *flat shading*. If we want the polygon to appear curved, we can assign different normals to each vertex and then OpenGL will interpolate the computed vertex colors across the polygon. This latter method is called *smooth* or *interpolative shading*. For objects composed of flat polygons, flat shading is more appropriate.

Let's again use the cube with the vertex numbering in **Figure 3**. We use the function `quad` to describe the faces in terms of the vertices

Listing 1: `quad.c`

```
GLfloat vertices[8][3] = {{-1.0,-1.0,-1.0}, {1.0,-1.0,-1.0},
    {-1.0,1.0,-1.0}, {1.0,1.0,-1.0},{-1.0,-1.0,1.0},
    {1.0,-1.0,1.0},{-1.0,1.0,1.0},{1.0,1.0,1.0}};

void quad(int a, int b, int c, int d)
{
    glBegin(GL_QUAD)
        glVertex3fv(vertices[a]);
        glVertex3fv(vertices[b]);
        glVertex3fv(vertices[c]);
        glVertex3fv(vertices[d]);
    glEnd();
}
```

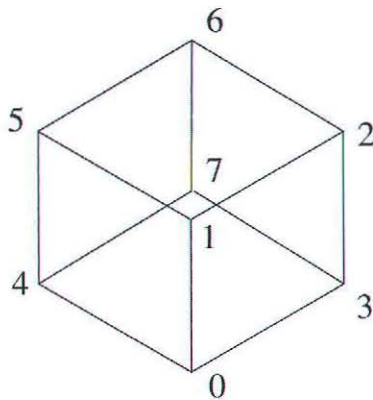


Figure 3. *Cube Vertex Labeling.*

To flat shade our cube, we make use of the six normal vectors, each of which points outward from one of the faces. Here is the modified `cube` function

Listing 2: Revised `cube.c` with normals for shading

```
GLfloat face_normals[6][3] = {{-1.0,0.0,0.0},{0.0,-1.0, 0.0},
    {0.0,0.0,-1.0},{1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0}};

void cube()
{
    glNormal3fv(face_normals[2]);
    quad(0, 2, 3, 1);
    glNormal3fv(face_normals[4]);
    quad(2, 6, 7, 3);
    glNormal3fv(face_normals[0]);
    quad(0, 4, 6, 2);
    glNormal3fv(face_normals[3]);
    quad(1, 3, 7, 5);
    glNormal3fv(face_normals[5]);
    quad(4, 5, 7, 6);
    glNormal3fv(face_normals[1]);
    quad(0, 1, 5, 4);
}
```

Now that we have specified the orientation of each face, we must describe the light source(s) and the material properties of our polygons. We must also enable lighting and the individual light sources. Suppose that we require just one

light source. We can both describe it and enable it within `myinit`. OpenGL allows each light source to have separate red, green and blue components and each light source consists of independent ambient, diffuse and specular sources. Each of these sources is configured in a similar manner. For our example, we will assume our cube consists of purely diffuse surfaces, so we need only worry about the diffuse components of the light source. Here is a `myinit` for a white light and a red surface:

Listing 3: Revised `myinit.c` with lights and materials

```
void myinit()
{
    GLfloat mat_diffuse[]={1.0, 0.0, 0.0, 1.0};
    GLfloat light_diffuse[]={1.0, 1.0, 1.0, 1.0};
    GLfloat light0_pos[4] = { 0.5, 1.5, 2.25, 0.0 };

    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);

    /* define material properties for front face of all polygons */

    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

    glEnable(GL_LIGHTING); /* enable lighting */
    glEnable(GL_LIGHT0); /* enable light 0 */
    glEnable(GL_DEPTH_TEST); /* Enable hidden—surface—removal */
    glClearColor(1.0, 1.0, 1.0, 1.0);
}
```

Both the light source and the material have RGBA components. The light source has a position in four-dimensional homogeneous coordinates. If the last component is one, then the source is a point source located at the position given by the first three components. If the fourth component is zero, the source is a distant parallel source and the first three components give its direction. This location is subject to the same transformations as are vertices for geometric objects. **Figure 4** shows the resulting image

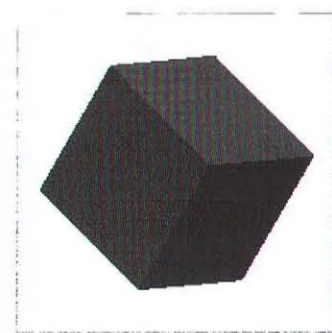


Figure 4. *Red Cube with Diffuse Reflections.*

TEXTURE MAPPING

While the capabilities of graphics systems are measured in the millions of shaded polygons per second that can be rendered, the detail needed in animations can require much higher rates. As an alternative, we can "paint" the detail on a

smaller number of polygons, much like a detailed label is wrapped around a featureless cylindrical soup can. Thus, the complex surface details that we see are contained in two-dimensional images, rather than in a three-dimensional collection of polygons. This technique is called texture mapping and has proven to be a powerful way of creating realistic images in applications ranging from games to movies to scientific visualization. It is so important that the required texture memory and mapping hardware are a significant part of graphics hardware boards.

OpenGL supports texture mapping through a separate pixel pipeline that processes the required maps. Texture images (arrays of texture elements or *texels*) can be generated either from a program or read in from a file. Although OpenGL supports one through four-dimensional texture mapping, to understand the basics of texture mapping we shall consider only two-dimensional maps to three-dimensional polygons as in **Figure 5**.

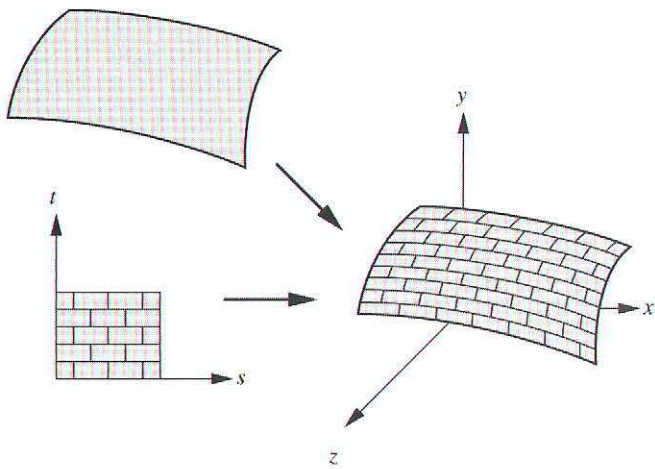


Figure 5. Texture Mapping a Pattern to a Surface.

We can regard the texture image as continuous with two-dimensional coordinates s and t . Normally, these coordinate range over $(0,1)$ with the origin at the bottom-left corner of the image. If we wish to map a texture image to a three-dimensional polygon, then the rasterizer must match a point on the polygon with both a point in the frame buffer and a point on the texture map. The first map is defined by the various transformations that we discussed in Part 1. We determine the second map by assigning texture coordinates to vertices and allowing OpenGL to interpolate intermediate values during rasterization. We assign texture coordinates via the function `glTexCoord` which sets up a present texture coordinate as part of the graphics state.

Consider the example of a quadrilateral. If we want to map the entire texture to this polygon, we can assign the four corners of the texture to the vertices

Listing 4: Assigning texture coordinates

```
glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0);
    glVertex3fv(a);
    glTexCoord2f(1.0, 0.0);
    glVertex3fv(b);
    glTexCoord2f(1.0, 1.0);
    glVertex3fv(c);
    glTexCoord2f(0.0, 1.0);
    glVertex3fv(d);
glEnd();
```

Figure 6 shows a checkerboard texture mapped to our cube. If we assign the texture coordinates over a smaller range, we will map only part of the texture to the polygon and if we change the order of the texture coordinates we can rotate the texture map relative to the polygon. For polygons with more vertices, the application program must decide on the appropriate mapping between vertices and texture coordinates, which may not be easy for complex three-dimensional objects. Although OpenGL will interpolate the given texture map, the results can appear odd if the texture coordinates are not assigned carefully. The task of mapping a single texture to an object composed of multiple polygons in a seamless manner can be very difficult, not unlike the real world difficulties of wallpapering curved surfaces with patterned rolls of paper.

Like other OpenGL features, texture mapping first must be enabled (`glEnable(GL_TEXTURE)`). Although texture mapping is a conceptually simple idea, we must also specify a set of parameters that control the mapping process. The major practical problems with texture mapping arise because the texture map is really a discrete array of pixels that often come from images. How these images are stored can be hardware and application dependent. Usually, we must specify explicitly how the texture image is stored (bytes/pixel, byte ordering, memory alignment, color components). Next we must specify how the mapping in **Figure 5** is to be carried out. The basic problem is that we want to color a point on the screen but this point when mapped back to texture coordinates normally does not map to an s and t corresponding to the center of a texel. One simple technique is to have OpenGL use the closest texel. However, this strategy can lead to a lot of jaggedness (*aliasing*) in the resulting image. A slower alternative is have OpenGL average a group of the closest texels to obtain a smoother result. These options are specified through the function `glTexParameter`. Another issue is what to do if the value of s or t is outside the interval $(0,1)$. Again using `glTexParameter`, we can either clamp the values at 0 and 1 or use the range $(0,1)$ periodically. The most difficult issue is one of scaling. A texel, when projected onto the screen, can be either much larger than a pixel or much smaller. If the texel is much smaller, then many texels may contribute to a pixel but will be averaged to a single value. This calculation can be a very time consuming and results in the color of only a single pixel. OpenGL supports a technique called *mipmapping* that allows a program to start with a single texture array and form a set of

smaller texture arrays that are stored. When texture mapping takes place, the appropriate array — the one that matches the size of a texel to a pixel — is used. The following code sets up a minimal set of options for a texture map and defines a checkerboard texture.

Listing 5: minimum texture map setup in myinit.c

```
void myinit()
{
    GLubyte image[64][64][3];
    int i, j, c;
    for(i=0;i<64;i++) for(j=0;j<64;j++)
    {
        /* Create an 8 x 8 checkerboard image of black and white texels */
        c = (((i&0x8)==0)^((j&0x8)==0))*255;
        image[i][j][0] = (GLubyte) c;
        image[i][j][1] = (GLubyte) c;
        image[i][j][2] = (GLubyte) c;
    }
    glEnable(GL_DEPTH_TEST); /*Enable hidden-surface removal*/
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glEnable(GL_TEXTURE_2D); /* Enable texture mapping */
    glTexImage2D(GL_TEXTURE_2D,0,3,64,64,0,GL_RGB,
        GL_UNSIGNED_BYTE, image); /* Assign image to texture */
    /* required texture parameters */
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,
        GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,
        GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
        GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
        GL_NEAREST);
}
```

First we create a 64 x 64 image to be used for our texture. We enable texture mapping and then pick `image` as the texture map. The other parameters in `glTexImage2D` give the size of the texture map, specify how it is stored and that it will be applied to the red, green and blue components. The four calls to `glTexParameterf` are required to specify how values of `s` and `t` outside (0,1) are to be handled (clamped) and that we are willing to use the nearest texel (for speed) rather than a filtered value. Figure 6 shows the cube with both texture mapping and shading. Note that in this mode texture mapping modifies the color determined by the shading calculation. Alternately, we can have the texture completely determine the color (*decating*).

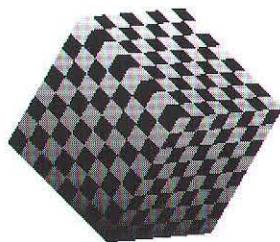


Figure 6. Texture Mapped Cube.

CLIENTS AND SERVERS

In many applications we must convey three-dimensional graphical information over a network, either to show the graphics remotely or to make use of hardware available on a remote machine. Web applications often fall into this category. Although we could send two-dimensional images across a network, the volume of data can be huge as compared to the compact description provided by three-dimensional geometry. Furthermore, in distributed interactive applications that involve manipulating large graphical databases, we would prefer to not have to send the database over the network repeatedly in response to small changes, such a change in viewing parameters.

In OpenGL, we regard the hardware with the display and the rendering engine as a graphics server and the program that defines and controls the graphics as a client. In what is called *immediate-mode* graphics, entities are sent to the graphics server as soon as they are defined in a program and there is no memory of these entities in the system. We used this mode in our sample cube programs. To redisplay the cube, we had to reexecute the code defining it. Thus, when we rotated the cube, we had to both alter the model-view matrix and reexecute the cube code defining its surfaces. In *retained-mode* graphics, graphical entities are defined and placed in structures called *display lists*, which are kept in the graphics server. For example, if we want to define a quadrilateral, store it on the server, and display it, we wrap its description between a `glBeginList` and a `glEndList`:

Listing 4: Display list for a quadrilateral

```
glBeginList(myQuad, GL_COMPILE_AND_DISPLAY);
    glBegin(GL_QUADS);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glVertex3fv(d);
    glEnd();
glEndList();
```

In this example `myQuad` is an integer identifier for our retained object and the flag `GL_COMPILE_AND_DISPLAY` indicates that we want to define the list and display it on the server. If we only want to place a display list on the server without rendering it, we use the flag `GL_COMPILE`. Most OpenGL functions can appear inside a display as can other code. Similarly, to put a cube on the server, we need only surround any of our previous cube code with a `glBeginList` and a `glEndList`. We might also use display lists to put a character set on the server. In general, any objects we intend to redisplay are good candidates for display lists.

Once a display list is on the server, we can have it rendered by the command such as

```
glExecuteList(myCube)
```


Suppose that we wish to rotate the object and then redisplay it such as in our rotating cube example from Part 1. Within the display callback we would see code such as

```
GlRotatef(axis_x, axis_y, axis_z, theta)
GlExecuteList(myCube);
```

In terms of the traffic between the client program and the graphics server, we would be sending only the rotation matrix and function calls but not the object, as it is already stored on the server. If we did the same example using a complex object with thousands of vertices, once the object was placed on the server, further manipulation would require no more network traffic than the manipulation of a single quadrilateral or our cube.

OpenGL plays a vital, but often invisible, role in three-dimensional Web applications using VRML (Virtual Reality Modeling Language). VRML is based on the Open Inventor data base model. Open Inventor is an object-oriented graphics system built on top of OpenGL's rendering capabilities. VRML applications are client-server based with the rendering done on the client end. Thus, a VRML browser must be able to render databases that contain geometry and attributes that looks like OpenGL entities. Consequently, an obvious way to build a VRML browser is as an OpenGL application where the VRML server places display lists on the graphics server.

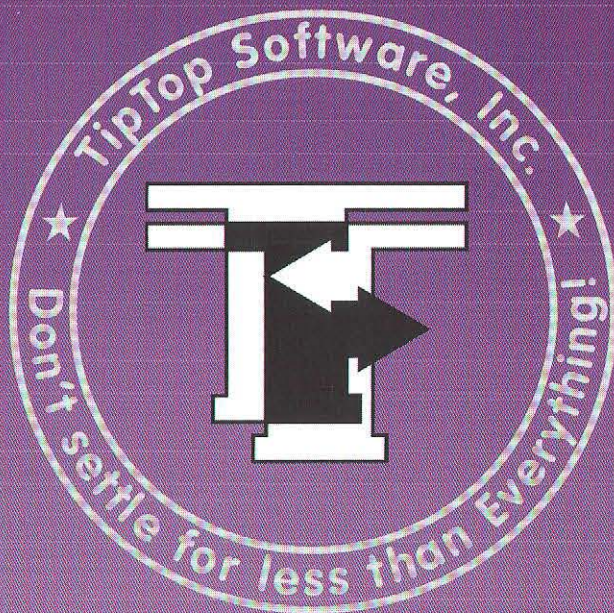
BUFFERS

OpenGL provides access to a variety of buffers. These include

- Color buffers (including the frame buffer).
- Depth Buffer.
- Accumulation Buffer.
- Stencil Buffer.

Many of these buffers have conventional uses, such as the frame buffer and the depth buffer, but all can be read from and written into by user programs and thus their uses are unlimited. In addition, the OpenGL architecture contains a variety of tables associated with these buffers and a variety of tests that can be performed on data as it is read from or written into buffers. Note that because these buffers are part of the graphics system, in an implementation in which these buffers and the associated tables are in separate hardware, once data have been moved to these buffers, we can achieve extremely high data rates as the system processor and bus are no longer involved.

For each type of buffer, we shall consider a few possible uses without going into coding details. We have seen that we can use multiple color buffers for double buffering. We can also use them for stereo viewing by rendering the left and right eye views into LEFT and RIGHT buffers and synching their display with special glasses that alternate presenting the images to the left and right eyes. For stereo animations, we can use four color buffers: FRONT_LEFT, FRONT_RIGHT, BACK_LEFT and BACK_RIGHT. More generally, we can use



Objective-Everything™

- A MUST-HAVE tool for MacOS X/Server!
- The preeminent scripting solution for Yellow Box.
- Rapid prototyping and exploratory programming as you have never seen it before!
- True language independence – use the best programming language for the task at hand, or even mix-and-match languages!
- Use Python, Tcl, Perl, WebScript, JavaScript, Java and Objective-C.
- Browse objects in a running program.
- Investigate program structure and object inter-relations.
- Class hierarchy, methods, instance variables, and other object information can be viewed and *edited* during program execution.
- **FREE for non-commercial use!**

WOEverything™

- Brings the power of Objective-Everything to WebObjects.
- Includes the popular WOPerl module!

www.tiptop.com

TipTop Software, Inc. ★ PO Box 30681 ★ Bethesda MD 20824 ★ USA
301-656-3837 ★ 301-656-8432 fax

color buffers that are not being displayed to enhance the power of the geometric pipeline. For example, shadows are projections of objects from the perspective of the light source. We can do an off screen rendering with the camera at the light source into one of the buffers and then carefully composite this image with the standard rendering. Another application is to render each object in a different color into an off-screen buffer. We can use the location of the mouse to point into this buffer and the color read at this location is an identifier for the object, a simple way of doing interactive object selection or *picking*.

The depth buffer is used in conjunction with many of the applications of the color buffers. One interesting use is for combining translucent and opaque polygons in a single scene. In our example of blending in Part 1, we turned off hidden-surface removal because all the polygons were translucent. If some of the polygons are opaque, unless the user program sends the polygons down the pipeline in the correct order, no combination of enabling hidden-surface removal and blending will produce a reasonable image. Consider what happens if we render opaque polygons first, and then make the depth buffer read-only for translucent polygons. Translucent polygons behind opaque polygons will be hidden, while those in front will be blended.

Color buffers typically have limited resolution, such as one byte per color component. Consequently, doing arithmetic calculations with color buffers can be subject to loss of color resolution. The accumulation buffer has sufficient depth that we can add multiple images into it without losing resolution. Obvious uses of such a buffer include image compositing and blending. To composite n images, we can add them individually into the accumulator buffer and then read out the result while scaling each color value by $1/n$. If we had tried to add these images into the frame buffer, we would have risked overflowing the color values, which are typically are stored 8 bits/component. If we tried to scale the colors before we added them into the frame buffer, we would have lost most, if not all, of our color resolution. For example, if $n=8$, we could lose three bits per color component. With an accumulator buffer, we can trade resolution for an increase in compositing time.

Less obvious, but easy to implement, applications of the accumulation buffer include digital filtering of images, scene antialiasing, depth of field images, and motion blur. Consider, for example, the antialiasing problem for polygons. As polygons are rasterized, the rasterizer computes small elements called *fragments* which are at most the size of a pixel. Each fragment is assigned a color that can determine the color of the corresponding pixel in the frame buffer. Generally, if the fragment is small or fragments from multiple polygons lie on the same pixel we will see jagged images if we make binary decisions as to whether or not a given fragment completely

determines the color of a pixel. One solution is to use the alpha channel we discussed to Part 1 to allow small amounts of color from multiple fragments to blend together. Unfortunately, this method can be very slow. An alternative is to render the scene multiple times into the accumulation buffer, each time with the viewer shifted very slightly. Each image will contain slightly different aliasing artifacts that will be averaged out by the accumulation process.

The stencil buffer allows us to draw pixels based on the corresponding values in the stencil buffer. Thus, we can create masks in the stencil buffer that we can use to do things such as creating windows into scenes or for placing multiple images in different parts of the frame buffer. What makes this buffer more interesting is that we can change its values as we render. This capability allows us to write programs that can determine if an object is in shadow or color objects differently if they are sliced by a plane.

PERFORMANCE TUNING

OpenGL supports a well-defined architecture and as such can be implemented in hardware, software or a combination of the two. Because there are both geometric and discrete pipelines, not only are there a wide range of implementation strategies, where bottlenecks arise depends on both the application and how the programmer chooses to use the OpenGL architecture. Consequently, it is difficult to make "one size fits all" solutions to performance issues. We can survey a few possibilities.

Defining geometric objects with polygons is simple but can lead to many function calls. For example, our cube with vertex colors, normals and texture coordinates required 108 OpenGL function calls: six faces each requiring a `glBegin` and `glEnd`, four vertices per face, each requiring a `glVertex`, `glColor`, `glNormal` and `glTexCoord`. One way to avoid this problem if the object is to be drawn multiple times is to use display lists. Another is to use *vertex arrays*, a feature added in OpenGL 1.1. In `myinit`, we can now set up and enable arrays that contain all the required information (colors, normals, vertices, texture coordinates). A single OpenGL call

```
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);
```

will render the six quadrilateral faces) whose indices are stored as unsigned bytes in the array `cubeIndices`, which contains the 24 vertices in the order they appear in our cube function above.

In the geometric pipeline lighting calculations can be very expensive, especially if there are multiple sources, as we have to do an independent calculation for each source. A large part of the computation is that of the vectors in the Phong model. If a polygon is small relative to the distance to a viewer, the view vector will not change significantly over the face of the polygon. If we use


```
GLLightModeli(LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE);
```

we allow the implementation to take advantage of this situation. We can also tell OpenGL whether we wish light calculations to be done on only one side of surface through

```
GLLightModeli(LIGHT_MODEL_TWO_SIDE, GL_FALSE);
```

We can also automatically eliminate (or *cull*) all polygons which are not facing the viewer by enabling culling

```
(GLCullFace (GL_BACK)).
```

Texture calculations can also be very time consuming and are subject to aliasing problems. OpenGL allows us to decide if we want to filter a texture to get a smoother image or just use the closest texel which is faster. Perspective projections can be a problem for texture mapping as the interpolation is more complex. In many situations, either the error is small enough that we do not care about it or the image is changing so rapidly that we cannot notice the error. In these situations, we can tell OpenGL not to correct for perspective.

More generally, we have to worry about both polygons and pixels, which are passing through two very different pipelines. Depending on both the implementation and the application, either pipeline can be the bottleneck. Often performance tuning, involves deciding which algorithm best matches the hardware and making creative use of the many features available in OpenGL. With OpenGL supported directly in the hardware of many new graphics cards, many graphics applications programmers are rethinking how to create images. For example, with the large amounts of texture memory included on these cards, in many situations we can generate details through textures rather than through geometry. Often, we can also avoid lighting calculations by storing some carefully chosen textures.

CONCLUSION

With over 200 functions in the API, we have only scratched the surface of what we can do with OpenGL. The major omission in these two articles is how we can define various types of curves and surfaces. Nevertheless, you should have a fair idea of the range of functionality supported by the OpenGL architecture. In the future, the CAD and animation communities will not only use OpenGL as their standard API but also start making use of features particular to OpenGL, such as the accumulation and stencil buffers.

The advantages of OpenGL are many. It is close to the hardware but still easy to use to write application programs. It is portable and supports a wide variety of features. It is the only graphics API that I have seen in my 15 years in the field that is used by animators, game developers, CAD engineers and researchers on supercomputers. Personally, I routinely use OpenGL on a PowerMac 6100, a PowerBook, an SGI

Infinite Reality Engine and a variety of PCs, rarely having to change my code moving among these systems. There is not much more that can I ask of an API.

SOURCES AND URLS

OpenGL is administered through an Architectural Review Board. The two major sources for on-line information on OpenGL are the OpenGL organization <<http://www.opengl.org>> and Silicon Graphics Inc <<http://www.sgi.com/Technology/OpenGL>>. You can find pointers to code, FAQ, standards documents and literature at these sites. I keep the sample code from my book at <<ftp://ftp.cs.unm.edu/pub/angel/BOOK>>.

OpenGL is available for most systems. For Mac users, there is an implementation from Conix Enterprises <<http://www.conix3d.com>> that includes support for GLUT and for hardware accelerators. There is a free OpenGL-like API called Mesa <<http://www.ssec.wisc.edu/~brianp/Mesa.html>> that can be compiled for most systems, including linux, and will run almost all OpenGL applications. There is a linux version available from Metro Link <<http://www.metrolink.com>>. You can obtain the code for GLUT and many examples at <<http://reality.sgi.com/opengl/glut3/glut3.html>>.

BIBLIOGRAPHY AND REFERENCES

- Angel, Edward. *Interactive Computer Graphics: A top-down approach with OpenGL*. Addison-Wesley, Reading, MA, 1997.
- Architectural Review Board. *OpenGL Reference Manual, Second Edition*, Addison-Wesley, 1997.
- Kilgard, Mark, *OpenGL Programming for the X Window System*, Addison-Wesley, 1996.
- Neider, Jackie, Tom Davis and Mason Woo. *The OpenGL Programming Guide, Second Edition*. Addison-Wesley, Reading, MA, 1997.
- Wright, Richard Jr. and Michael Sweet, *The OpenGL Superbible*, Waite Group Press, Corte Madera, CA, 1997.

ACKNOWLEDGEMENTS

I would like to thank Apple Computer, Silicon Graphics, and Conix Enterprises for the hardware and software support that enabled me to write my OpenGL textbook.

MT

Interested in writing for the
magazine? Contact us about a
writer's kit at
<<mailto:editorial@mactech.com>>

by Brent Schorsch

Inside InputSprocket

How to Use InputSprocket to Support User Input in Games

INTRODUCTION

Handling user input is a necessary aspect of a game. Without user input, it is not a game, but just a flashy movie. This article will cover what a game developer needs to do in order to handle user input using InputSprocket, Apple's gaming input API. InputSprocket was created to address the lack of a common API for joysticks and other gaming devices on the Macintosh platform. With the introduction of the iMac and USB, InputSprocket has opened the door for numerous gaming hardware manufactures to sell their products to Macintosh customers. Game developers also benefit by supporting InputSprocket because their game will support the latest input gadgets with no extra effort from the game developer. Game players benefit from the increased options.

THE BASICS

InputSprocket is a relatively simple API to use. There are some more complicated aspects, which will be covered later, but the basics are straightforward. First, an application must provide a list of its input **needs** to InputSprocket. Example needs a game

might have are: fire weapon, jump, turn, look, rudder, lower landing gear. The list of needs determines how the user can interact with the game and therefore will directly affect whether the game is thought to be hard to control or very intuitive and flexible.

Once InputSprocket is initialized and has the list of needs, there is a single API call (ISpConfigure) which will bring up a user interface, provided by InputSprocket, for the user to map the physical **elements** of the device to needs in the game. InputSprocket maintains the user's settings in a preference file. This dialog is pictured in **Figure 1**.

During game play, the application can either get events or poll the current value for any of these needs. Typically, applications will get events on 'button kind' needs (and others) and poll the value of 'axis kind' needs once per game loop. The different 'kinds' of needs is covered later.

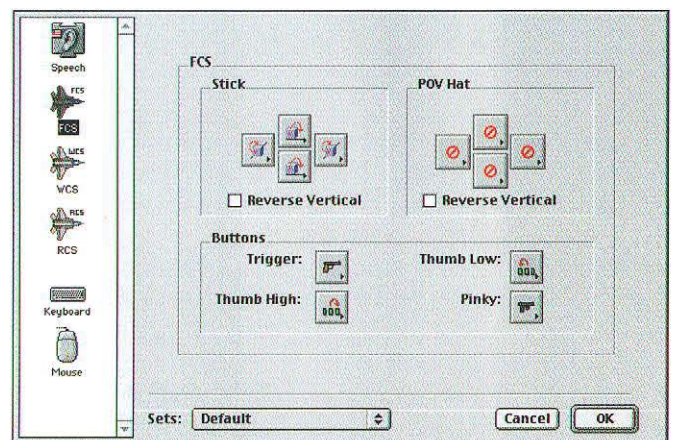


Figure 1. InputSprocket Configure Dialog.

Brent Schorsch is the engineer at Apple Computer, Inc. responsible for InputSprocket. Brent enjoys reading science fiction novels, reading a dozen or so books a month. In addition, Brent is an avid game enthusiast. His favorite games are those that can be played against human opponents. In such battles, he answers to the handle 'Ender' which he acquired playing Bungie's 'Minotaur' back in the dark ages. You might find him online on iMagicOnline's WWII flight-sim, Warbirds, using the handle 'endr-' flying for purple.

THE 'LOW LEVEL' INTERFACE

In fact, there are two different ways an application can use InputSprocket: 'low-level' and 'high-level'. The 'low-level' interface provides a means for the developer to get a list of the devices currently available and manually check the state of each device. This interface is strongly discouraged. It is provided for those developers who wish to provide their own user interface to configuring device, but it puts a much greater burden on the game developer. With this interface, the game developer is responsible for determining how each kind of device might be configured to work with their game and to provide a complete user interface to make this configuration. Games that use the 'low-level' interface lose any extra functionality that is provided by the high level interface. The rest of this article will focus on the 'high-level' interface, which is the one that all game developers are encouraged to use.

DETERMINING THE GAME'S NEEDS

There are four basic need kinds: **button**, direction pad (**dpad**), **axis** and **delta**. (There is also a movement kind, but its use is discouraged.) Each kind has a different data format. The button kind is the simplest, with values of up or down. A dpad has nine possible values: idle, left, up-left, up, up-right, right, down-right, down, and down-left. An axis kind's value is a 32 bit unsigned number, regardless of whether it is symmetric. Finally, the value of a delta type is a 32 bit Fixed point number of inches moved.

When deciding what kind to make a particular need, one should prefer the axis kind to button and dpad kinds when applicable. This will allow game players to get the full functionality out of their equipment. Sure, keyboard users may only be able to either walk or run (forwards or backwards), but if someone has a device with enough 'analog' inputs, the game should let him continuously vary his speed. In some cases, it may be necessary to provide some extra information to InputSprocket so that the keyboard user experience is maintained which is covered later. Delta kinds are provide data similar to 'raw mouse' data but due to limitations in the current InputSprocket drivers, are most useful in the more complex InputSprocket cases, where once again extra information is provided.

In some cases, it may make sense to provide several needs that affect the same thing in the game world. For example, in addition to a 'next weapon' and 'previous weapon' need, it often makes sense to provide needs to switch to specific weapons, e.g. 'use machine gun', 'use plasma rifle'. The code necessary to support these different means to switch weapons is trivial, but the added functionality to certain users (with the right hardware) is immense. The speech recognition InputSprocket driver does not add much if you can only say 'next weapon', but if you can say 'use plasma rifle', then it might start to be appealing. Another case where multiple needs might make sense is where the need is some form of toggle, such as 'map mode'

or 'landing gear'. One button may be sufficient for most users, but if the hardware physically has two states, like the caps lock key on some keyboards or any sort of lever, then it is very easy for the physical device to become out of sync with the game. However, by adding two more needs, switch to map view (or landing gear up) and switch to non-map view (or landing gear down), it is impossible, when properly configured, for the hardware to be out of sync with the game, regardless of the starting state of the game and the hardware.

Finally, it often makes sense to use the native data types as input for the game. This is a case where the game essentially gives InputSprocket extra information so that it can behave better. It does this by providing multiple needs, of different data kinds, to change the same thing in the game state. An example where this is useful is for the traditional rudder controls on a flight-sim. Typically, the keyboard commands to change the rudder are 'Increase Left Rudder', 'Center Rudder', and 'Increase Right Rudder'. Each time either of the 'Increase ...' keys are pressed, the current rudder value is changed. In order to restore the plane back to no rudder, you have to either manually press the keys enough times to get back to center, or press the 'Center Rudder' key. The current 'InputSprocket Keyboard' driver does not allow the user to configure an axis need to three keys like this, so this case calls for using extra needs. In addition to the rudder axis need (for those who have a device such as a rudder or twisting joystick), the game will have three button needs: left rudder, center rudder, and right rudder. The axis need should set the `kISpNeedFlag_Axis_AlreadyButton` bit in the `ISpNeed`, flags field and the button needs should set the `kISpNeedFlag_Button_AlreadyAxis` bit. Another common case for using button and axis types is for a throttle in a flight-sim.

Listing 1 contains the complete list of needs chosen for the sample application. **Listing 2** is an excerpt from the sample application which demonstrates using axis and button types to change yaw angle. The `kISpNeedFlag_Button_AlreadyDelta` bit is also set, because the sample application also reads delta types separately. Typically, delta types are used when the developer wants the 'feel' of using a mouse to be similar to how it typically feels in a first person shooter, like Unreal or Quake.

Listing 1: ISp_Sample.h

This is an excerpt from ISp_Sample.h containing the enumeration of all the needs.

```
enum
{
    // primary needs

    kNeed_FireWeapon,
    kNeed_NextWeapon,
    kNeed_PreviousWeapon,

    kNeed_Roll,
    kNeed_Pitch,
    kNeed_Yaw,
    kNeed_Throttle,
```



```

kNeed_StartPause,
kNeed_Quit,

// secondary needs for alternative input kinds

kNeed_Weapon_MachineGun,
kNeed_Weapon_Cannon,
kNeed_Weapon_Laser,
kNeed_Weapon_Missile,
kNeed_Weapon_PrecisionBomb,
kNeed_Weapon_ClusterBomb,

kNeed_Roll_AsDelta,
kNeed_Pitch_AsDelta,
kNeed_Yaw_AsDelta,

kNeed_Yaw_Left,
kNeed_Yaw_Center,
kNeed_Yaw_Right,

kNeed_Throttle_Min,
kNeed_Throttle_Decrease,
kNeed_Throttle_Increase,
kNeed_Throttle_Max,

kNeed_NeedCount
};

```

Listing 2: ISp_Sample.c

Input_Initialize

This is an excerpt from Input_Initialize which initializes the needs array.

```

//• we'll init all the player 1 items now
//• (everything but quit unless we add a second player)
tempNeed.playerNum = 1;
// [snip - code removed from this listing]
// Now group 4, which is for changing yaw
tempNeed.group = 4;

GetIndString (tempNeed.name, kSTRn_NeedNames,
              kNeed_Yaw + 1);
tempNeed.iconSuiteResourceId = 1000 + kNeed_Yaw;
tempNeed.theKind = kISpElementKind_Axis;
tempNeed.theLabel = kISpElementLabel_Axis_Yaw;
tempNeed.flags = kISpNeedFlag_Axis_AlreadyButton;
myNeeds[kNeed_Yaw] = tempNeed;

GetIndString (tempNeed.name, kSTRn_NeedNames,
              kNeed_Yaw_Left + 1);
tempNeed.iconSuiteResourceId = 1000 + kNeed_Yaw_Left;
tempNeed.theKind = kISpElementKind_Button;
tempNeed.theLabel = kISpElementLabel_None;
tempNeed.flags = kISpNeedFlag_Button_AlreadyAxis |
                 kISpNeedFlag_Button_AlreadyDelta |
                 kISpNeedFlag_EventsOnly;
myNeeds[kNeed_Yaw_Left] = tempNeed;

GetIndString (tempNeed.name, kSTRn_NeedNames,
              kNeed_Yaw_Center + 1);
tempNeed.iconSuiteResourceId = 1000 + kNeed_Yaw_Center;
tempNeed.theKind = kISpElementKind_Button;
tempNeed.theLabel = kISpElementLabel_None;
tempNeed.flags = kISpNeedFlag_Button_AlreadyAxis |
                 kISpNeedFlag_Button_AlreadyDelta |
                 kISpNeedFlag_EventsOnly;
myNeeds[kNeed_Yaw_Center] = tempNeed;

GetIndString (tempNeed.name, kSTRn_NeedNames,
              kNeed_Yaw_Right + 1);
tempNeed.iconSuiteResourceId = 1000 + kNeed_Yaw_Right;
tempNeed.theKind = kISpElementKind_Button;
tempNeed.theLabel = kISpElementLabel_None;
tempNeed.flags = kISpNeedFlag_Button_AlreadyAxis |
                 kISpNeedFlag_Button_AlreadyDelta |
                 kISpNeedFlag_EventsOnly;
myNeeds[kNeed_Yaw_Right] = tempNeed;

```

Listing 2 also demonstrates the other fields that must be filled in the ISpNeed structure. The name field contains a

string to be displayed to the user in the configuration dialog (shown in **Figure 1**), typically in the popup menus for each device element (although the strings appear directly for keyboard devices). The iconSuiteResourceId field contains the resource ID for an icon family that represents that need. Typically 'ics#' and 'ics8' icons are provided, since all the current drivers only display 16x16 icons. The theKind field determines whether the need is a button, delta, axis, dpad, or something else. The theLabel field is used to give hints to InputSprocket about how the need is used. There will inevitably be needs in every game that are not described by one of the labels in InputSprocket.h, those needs should use the kISpElementLabel_None label. Another bit set in the flags field in this listing is kISpNeedFlag_EventsOnly which tells InputSprocket that it does not have to maintain state information for this need. The group was set at the top of the listing, so that it is the same for all these needs. By convention, all the needs which affect the same game state are set to the same group. The full source for Input_Initialize in ISp_Sample.c uses five groups: changing weapon, roll, pitch, yaw, throttle.

INITIALIZATION

There are several InputSprocket functions which will typically be necessary during initialization. First, ISpGetVersion should be used to confirm that InputSprocket is available and of a minimum version for the game. Next, ISpStartup should be called to get InputSprocket up and running. ISpElement_NewVirtualFromNeeds is used to create and allocate virtual elements for each need. Virtual elements are the objects that are used to get events or are polled. Now ISpInit can be called to initialize the 'high-level' interface to InputSprocket. ISpInit provides InputSprocket with the needs list and the previously allocated virtual elements. It is often convenient to get events on an element list, which is just a grouping of (in this case virtual) elements. ISpElementList_New is used to allocate a new elements list and ISpElementList_AddElements is used to add one or more elements. Typically, one element list is used for all the regular button needs and an additional element list is created for each group of buttons which correspond to an axis need. **Listing 3** is an excerpt from Input_Initialize which demonstrates all of these functions.

Listing 3: ISp_Sample.c

Input_Initialize

This is an excerpt from Input_Initialize which initializes InputSprocket with the needs list and builds an element list.

```

//• Alright, now that the array is set up, we can call ISp to init stuff
err = ISpStartup ();
if (err)
    ErrorAlert("\pCould not Initialize InputSprocket.", err, true);

//• Setup the input sprocket elements
err = ISpElement_NewVirtualFromNeeds(kNeed_NeedCount,
                                     myNeeds, gInputElement, 0);
if (err)
    ErrorAlert("\pCould not create ISp virtual controls from needs.",
               err, true);

```



```

//• Init InputSprocket and tell it our needs
err = ISpInit (kNeed_NeedCount, myNeeds, gInputElement,
               kISpSampleCreator, kISpSampleNeedsVersion,
               0, ksetl_ISpSample, 0);
if (err)
    ErrorAlert("\pCould not initialize high-level ISp.", err, true);

//• Create a element list containing all the 'normal' buttons (we get events on these)
err = ISpElementList_New(0, NULL, &gEventsElementList, 0);
if (err)
    ErrorAlert("\pCould not create button element list.", err, true);

//• we set the refcon to the need enum value, so we can use it later
//• doing some shortcut error checking for readability
err = ISpElementList_AddElements (gEventsElementList,
                                   kNeed_FireWeapon, 1,
                                   &gInputElement[kNeed_FireWeapon]);
err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_StartPause, 1,
                                   &gInputElement[kNeed_StartPause]);

err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_NextWeapon, 1,
                                   &gInputElement[kNeed_NextWeapon]);
err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_PreviousWeapon, 1,
                                   &gInputElement[kNeed_PreviousWeapon]);

err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_Weapon_MachineGun, 1,
                                   &gInputElement[kNeed_Weapon_MachineGun]);
err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_Weapon_Cannon, 1,
                                   &gInputElement[kNeed_Weapon_Cannon]);
err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_Weapon_Laser, 1,
                                   &gInputElement[kNeed_Weapon_Laser]);
err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_Weapon_Missile, 1,
                                   &gInputElement[kNeed_Weapon_Missile]);
err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_Weapon_PrecisionBomb, 1,
                                   &gInputElement[kNeed_Weapon_PrecisionBomb]);
err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_Weapon_ClusterBomb, 1,
                                   &gInputElement[kNeed_Weapon_ClusterBomb]);

err |= ISpElementList_AddElements (gEventsElementList,
                                   kNeed_Quit, 1,
                                   &gInputElement[kNeed_Quit]);

if (err)
    ErrorAlert(
        "\pCould not fill button element list. Error number may be inaccurate.",
        err, true);

```

COOPERATING WITH MAC OS

By default, InputSprocket assumes that a game has its own method, using traditional Mac OS techniques, to get user input from all mice and keyboards. If a game wants to use InputSprocket for keyboard and/or mouse input, it must enable those **classes** of device with `ISpDevices_ActivateClass`. Activating the mouse class will disconnect all mice and trackballs from controlling the cursor, so should only be done while the game is in progress and only if the Mac OS cursor is not necessary to play the game. Activating the keyboard class will prevent the keyboard from generating any Mac OS events, although `GetKeys` will still function.

The game developer must decide whether to use InputSprocket for mouse and/or keyboard input. Using InputSprocket for mouse input is recommended for games that do not use the Mac OS cursor during play. The primary

advantages in this case are that multi-button mice are easily supported and configured, and that the user interface is consistent with other gaming devices. For those game developers who do not already have a method they prefer to get keyboard data, using InputSprocket for this purpose will save time. However, InputSprocket is not an appropriate way to get 'typing' input from the user (such as a message that the user is sending to other players), so the keyboard class should be deactivated and traditional Mac OS means used whenever 'typing' is initiated.

Because InputSprocket assumes control of all the active devices when it is active, it is necessary to suspend it when the application is placed in the background. When the application is placed in the foreground again, InputSprocket may be resumed. It is very important that InputSprocket not be left active when the application is not front-most. These operations are normally performed on the response of a suspend/resume OS event and are performed by the functions `ISpSuspend` and `ISpResume`. It is safe to use these functions at other times while the application is front-most if desired. However, a slightly better experience may be possible if just the keyboard and mouse classes are disabled. This way, for example, the 'start' button on a gamepad can still be used to start a game. The sample application takes the latter approach.

Bugs!
Hate 'em, right?
But what can you do about it?

BugLink_{solo}

- Single user bug tracking database
- Fully customizable to fit your needs
- Includes **BugLink Reporter**, a reporting tool you can redistribute freely with your software

BugLink

- Multi-user bug tracking database
- Includes everything in BugLink Solo
- TCP/IP architecture allows users to coordinate efforts around the world.

Demo Available at our web site
 BugLink Solo: \$99.95
 BugLink: \$299.95 for 5 users

MACWORLD SHOW SPECIAL!
 Buy BugLink Solo for \$59.95,
 a \$40 savings! Or try BugLink
 for \$199.95, a \$100 savings!

The PandaWave

<http://www.pandawave.com>

GETTING USER EVENTS

The sample application sets up a element list for all the normal button elements with the refcon value the same as the enum value for that need. This makes taking action based on the event almost trivial, as **Listing 4** demonstrates. `ISpElementList_GetNextEvent` is used to get the next event on the queue for the element list. `ISpTickle` is a way to give time to `InputSprocket` drivers even if the game does not call `WaitNextEvent`. It must be called at task level (not interrupt level), but is only required for drivers which must be manually enabled (like speech recognition). It is recommended that all games give time, but it is not necessary. It is also reasonable for a game to limit calls to `ISpTickle` to a few per second.

Listing 4: ISp_Sample.c

Input_GetButtonEvents
This function is used to modify the `gameState` structure if any of the primary buttons have been pressed. This function is called as part of the normal game loop.

```
void Input_GetButtonEvents (Input_GameState * gameState)
{
    OSStatus      error = noErr;
    ISpElementEvent event;
    Boolean        wasEvent;

    // give time to some non-interrupt driven input drivers (like speech recognition)
    ISpTickle ();

    // get all pending events
    do
    {
        error = ISpElementList_GetNextEvent (gEventsElementList,
                                             sizeof (event), &event, &wasEvent);

        if (wasEvent && !error)
        {
            switch (event.refCon)
            {
                case kNeed_FireWeapon:
                    if (event.data == kISpButtonDown)
                    {
                        gameState->fireWeaponState = true;
                        gameState->fireWeaponCount++;
                    }
                    else // (event.data == kISpButtonUp)
                        gameState->fireWeaponState = false;
                    break;

                case kNeed_StartPause:
                    if (event.data == kISpButtonDown)
                    {
                        if (!gameState->gameInProgress)
                            gameState->gameInProgress = true;
                        else
                            gameState->gamePaused =
                                !gameState->gamePaused;
                    }
                    break;

                case kNeed_NextWeapon:
                    if (event.data == kISpButtonDown)
                    {
                        gameState->currentWeapon++;
                        if (gameState->currentWeapon >=
                            kWeapon_WeaponCount)
                            gameState->currentWeapon = 0;
                    }
                    break;

                case kNeed_PreviousWeapon:
                    if (event.data == kISpButtonDown)
                    {
                        gameState->currentWeapon--;
                    }
            }
        }
    } while (wasEvent && !error);
}
```

```
        if (gameState->currentWeapon < 0)
            gameState->currentWeapon =
                kWeapon_WeaponCount - 1;
    }
    break;

case kNeed_Weapon_MachineGun:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_MachineGun;
    break;

case kNeed_Weapon_Cannon:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_Cannon;
    break;

case kNeed_Weapon_Laser:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_Laser;
    break;

case kNeed_Weapon_Missile:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_Missile;
    break;

case kNeed_Weapon_PrecisionBomb:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon =
            kWeapon_PrecisionBomb;
    break;

case kNeed_Weapon_ClusterBomb:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_ClusterBomb;
    break;

case kNeed_Quit:
    gameState->gameInProgress = false;
    break;
}
} while (wasEvent && !error);
}
```

POLLING AXIS VALUES

The sample application reads axis values for roll, pitch, yaw, and throttle. The most complicated one is yaw, which is shown in **Listing 5**. When there are both an axis and button needs that map to the same game state, there is an easy technique to get the correct behavior. First, `ISpElement_GetNextEvent` is used **on the axis element** just to check to see if the axis value changed. If it did change, then `ISpElement_GetSimpleState` is used to poll the current value of that axis element. Both the axis element and the element list are then flushed. However, if the axis value has not changed (`wasEvent` is false), then `ISpElementList_GetNextEvent` is used to get all the button events on the element list containing the buttons for the axis. Delta values are actually handled differently at a higher level, so they are read and accumulated into a separate value in the state structure.

Listing 5: ISp_Sample.c

Input_GetYaw
This function is used to modify the `gameState` structure to include the latest changes in yaw based on user input. This function is called as part of the normal game loop.

```
void Input_GetYaw (Input_GameState * gameState)
{
    OSStatus      error = noErr;
    ISpElementEvent event;
    Boolean        wasEvent;
    ISpAxisData    axisValue;
    SInt32         yawValue = gameState->yawInput;

    if (gameState->currentWeapon < 0)
        gameState->currentWeapon =
            kWeapon_WeaponCount - 1;
    break;

case kNeed_Weapon_MachineGun:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_MachineGun;
    break;

case kNeed_Weapon_Cannon:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_Cannon;
    break;

case kNeed_Weapon_Laser:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_Laser;
    break;

case kNeed_Weapon_Missile:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_Missile;
    break;

case kNeed_Weapon_PrecisionBomb:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon =
            kWeapon_PrecisionBomb;
    break;

case kNeed_Weapon_ClusterBomb:
    if (event.data == kISpButtonDown)
        gameState->currentWeapon = kWeapon_ClusterBomb;
    break;

case kNeed_Quit:
    gameState->gameInProgress = false;
    break;
}
} while (wasEvent && !error);
}
```



```
// we check the axis, to see if it was moved, if so, we use that value
error = ISpElement_GetNextEvent (gInputElement[kNeed_Yaw],
    sizeof (event), &event, &wasEvent);
if (!error && wasEvent)
{
    // we wish to ignore all button presses prior to this moment
    ISpElementList_Flush(gYawElementList);

    // get the current value
    error = ISpElement_GetSimpleState
        (gInputElement[kNeed_Yaw], &axisValue);
    if (!error)
        yawValue =
            ISpAxisToSampleAxis (axisValue, kMin_Yaw, kMax_Yaw);

    ISpElement_Flush(gInputElement[kNeed_Yaw]);
}
// otherwise, we check to see if one of the yaw buttons was pressed
else do
{
    error = ISpElementList_GetNextEvent (gYawElementList,
        sizeof (event), &event, &wasEvent);

    // only process valid keydown events (all the yaw events ignore button ups)
    if (wasEvent && !error && (event.data == kISpButtonDown))
    {
        switch (event.refCon)
        {
            case kNeed_Yaw_Left:
                yawValue -= kIncrement_Yaw;
                if (yawValue < kMin_Yaw) yawValue = kMin_Yaw;
                break;
            case kNeed_Yaw_Center:
                yawValue = kMin_Yaw + ((kMax_Yaw - kMin_Yaw) / 2);
                break;
            case kNeed_Yaw_Right:
                yawValue += kIncrement_Yaw;
                if (yawValue > kMax_Yaw) yawValue = kMax_Yaw;
                break;
        }
    }
}
```

```

    }
}
while (wasEvent && !error);

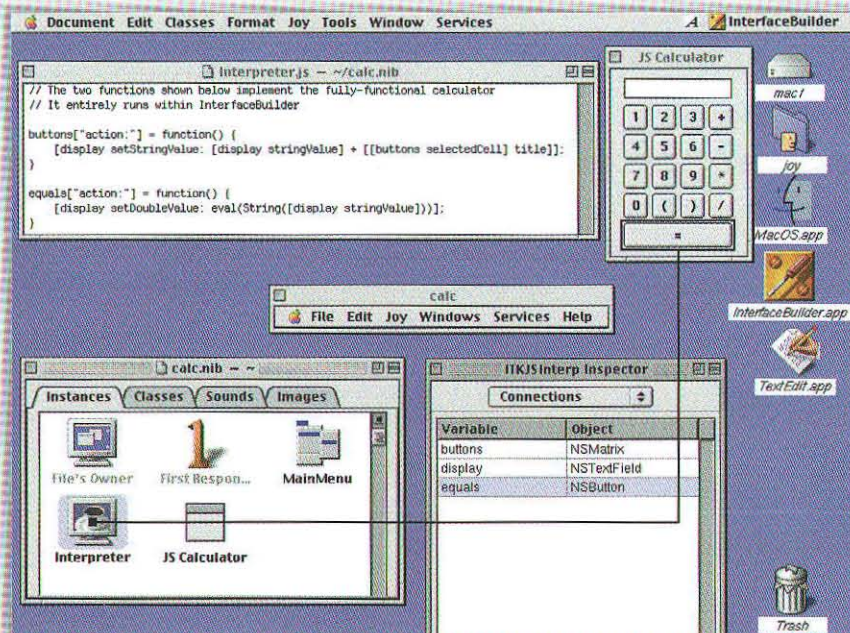
gameState->yawInput = yawValue;

//• also check the delta values
gameState->deltaYaw = 0;
do
{
    error = ISpElement_GetNextEvent
        (gInputElement[kNeed_Yaw_AsDelta],
            sizeof (event), &event, &wasEvent);
    if (wasEvent && !error)
        gameState->deltaYaw += (Fixed) event.data;
}
while (wasEvent && !error);
}
```

INPUTSPROCKET RESOURCES

InputSprocket checks for certain resources inside an application. First, the application should have an InputSprocket application resource ('isap') which simply specifies how the application uses InputSprocket. Next, the application should have a set list resource ('setl'). It is ok if there are no sets contained in this resource, but it should be present. The set list resource contains pointers to individual saved-set resources ('tset'). Saved set resources contain a set of configuration information for a single device for a specific application. InputSprocket.r contains a resource template for a saved-set resource that corresponds to a keyboard device. This template can be used to put keyboard

What a Joy!



Use the Joy-enhanced Interface Builder to create applications within minutes. Like this simple calculator. Design the GUI. Wire up the objects. Override two methods with Joy one-liners. It's easy! Run it in Interface Builder test mode with a \$69 Joy Explorer license. Get a \$399 Joy Developer license (\$339 when upgrading from Explorer); then, select "Save nib as app", select your platforms (all Yellow platforms supported) and have Joy save your file as an application that runs on all those platforms — size: only about 50k per platform!

Turbo-charge your OS X Development !

- ✦ **Not YASL**
Joy 2.0 is not Yet Another Scripting Language. Joy unifies JavaScript, C, Objective-C, and Java.
- ✦ **Look into frameworks**
Inspect classes and objects. Look at what methods are provided, and what instance variables there are.
- ✦ **Snoop into applications**
Take a look at what's going on inside an application. Even one for which you don't have the source.
- ✦ **Interact**
Double click on an object, double click on a method, hit return and it's executed!
- ✦ **Prototype**
Try out commands, then capture them in a method. Prototype in Joy, then compile (if necessary) for speed.
- ✦ **Use Joy for stress relief**
Joy takes the stress out of deploying to Windows. Run Joy on OS X, then save your project for Windows.

Price: \$ 69 / \$ 399

Available now for OS X Server, WebObjects and OPENSTEP.

Joy 1.0 was cool, Joy 2.0 is even cooler:

Joy 1.0 received a MacTech Best Tool for New Technologies Finalist Award. NEXTTOYOU called Joy 1.0 an afterburner for your software development.



defaults in a '.r' file. The saved set resources for all the other devices do not have templates. This means that the developer must execute the built game, manually create each set he wants to include with the game, and then copy these sets from the active 'Input Sprocket Preferences' file (in the Preferences folder) to his '.r' file. The sample application includes two sets for the keyboard and two for the mouse. Fortunately the default settings for most drivers (other than the keyboard) are pretty good if the developer is careful about ordering his needs list, so it is usually not necessary to provide sets besides those for the keyboard. **Listing 6** contains the resource description for the default keyboard set in the sample application. If the keyboard saved set resource does not have the correct number of items, then it will not function even though it may still be displayed in the sets menu. The easiest way to debug this is to compare a saved set resource generated by a '.r' file with one created by the 'InputSprocket Keyboard' driver in the 'InputSprocket Preferences' file. Also note that the default sets are only copied once from the application to the 'InputSprocket Preferences' file, so it usually will be necessary to delete the 'InputSprocket Preferences' file from the preferences folder several times during development.

Listing 5: ISp_Sample.r

```

                                ktset_DefaultKeyboard
This is the resource description for the default keyboard set in the sample application.

#define      rNoModifiers rControlOff, rOptionOff, rShiftOff, \
              controlOff, optionOff, shiftOff, commandOff

resource 'tset' (ktset_DefaultKeyboard, "Default (Keyboard)")
{
    supportedVersion,
    {
        /* kNeed_FireWeapon */
        kpd0Key, rNoModifiers,

        /* kNeed_NextWeapon */
        kpd9Key, rNoModifiers,

        /* kNeed_PreviousWeapon */
        kpd7Key, rNoModifiers,

        /* kNeed_Roll */
        /* min (left/down/back) */
        kpd4Key, rNoModifiers,

        /* max (right/up/forward) */
        kpd6Key, rNoModifiers,

        /* kNeed_Pitch */
        /* min (left/down/back) */
        kpd5Key, rNoModifiers,

        /* max (right/up/forward) */
        kpd8Key, rNoModifiers,

        /* kNeed_Yaw */
        /* this need does not generate any items, because it has later button equivalents */
        /* kNeed_Throttle */
        /* this need does not generate any items, because it has later button equivalents */

        /* kNeed_StartPause */
        tildeKey, rNoModifiers,

        /* kNeed_Quit */
        qKey, rControlOff, rOptionOff, rShiftOff,
        controlOff, optionOff, shiftOff, commandOn,

        /* kNeed_Weapon_MachineGun */
        nlKey, rNoModifiers,

```

```

        /* kNeed_Weapon_Cannon */
        n2Key, rNoModifiers,
        /* kNeed_Weapon_Laser */
        n3Key, rNoModifiers,
        /* kNeed_Weapon_Missile */
        n4Key, rNoModifiers,
        /* kNeed_Weapon_PrecisionBomb */
        n5Key, rNoModifiers,
        /* kNeed_Weapon_ClusterBomb */
        n6Key, rNoModifiers,

        /* kNeed_Roll_AsDelta */
        /* this need does not generate any items - the keyboard does not do deltas */
        /* kNeed_Pitch_AsDelta */
        /* this need does not generate any items - the keyboard does not do deltas */
        /* kNeed_Yaw_AsDelta */
        /* this need does not generate any items - the keyboard does not do deltas */

        /* kNeed_Yaw_Left */
        aKey, rNoModifiers,
        /* kNeed_Yaw_Center */
        sKey, rNoModifiers,
        /* kNeed_Yaw_Right */
        dKey, rNoModifiers,

        /* kNeed_Throttle_Min */
        kpdEqualKey, rNoModifiers,
        /* kNeed_Throttle_Decrease */
        kpdMinusKey, rNoModifiers,
        /* kNeed_Throttle_Increase */
        kpdPlusKey, rNoModifiers,
        /* kNeed_Throttle_Max */
        kpdSlashKey, rNoModifiers,
    }
};
};

```

FINAL WORD

Using InputSprocket for user input in games is a good idea. It is relatively simple to implement, and both the developer and game player benefit. With the introduction of the iMac, and its USB ports, the number of input devices available to Macintosh customers is ballooning. At the time this was written, the current version of InputSprocket was 1.4. The latest version and information is available at <<http://developer.apple.com/games/sprockets>>. Examining the sample application, which was written using the techniques in this article, should be the next step for a developer interested in using InputSprocket. The finished application is pictured in **Figure 2**.

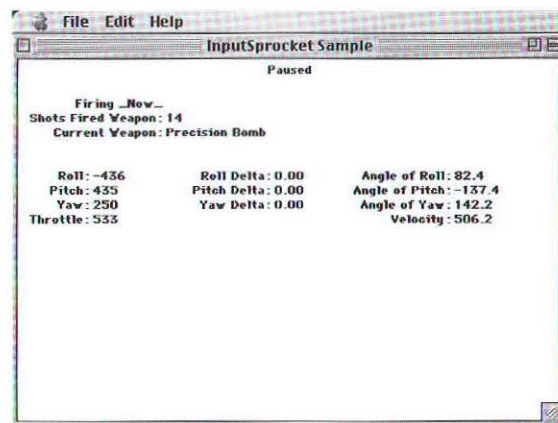


Figure 2. ISp_Sample in action.

by David McKee

Edited by Michael Brian Bentley

External Function Plug-ins for FileMaker Pro

An Introduction to Their Nature and Creation.

FILEMAKER PRO AND PLUG-INS

FileMaker® Pro software is the database of choice for Mac OS users; part of its success has been due to the very accessible user interface. Much of what you do is point-and-click. Up until recently, the closest to programming you could get is a very complex calculation formula or script. Plug-ins introduce a level of programmability never before available.

A script in FileMaker Pro is not unlike a macro in a word processing application. Scripts are great for automating tasks you can do manually in the user interface such as switching layouts or printing a report, but they must be initiated by the user. Calculations are performed without user input and are adequate for typical usage, since calculations are written in a high level syntax that is interpreted by the FileMaker Pro calculation engine. Creating a formula merely requires that you assemble a series of built-in functions from various categories.

A calculation formula like "fieldA + Sin(fieldB)" can be created entirely by the point-and-click interface.

The problem is, formulas that require recursion or looping are not possible in calculations. Typical kinds of functions are built-in, like trigonometric functions, but industry-specific

algorithms, like annuity functions based on payments at the beginning of a period, are not.

Scripts support recursion and looping, but require the user to execute them.

Both calculations and scripts interact with other parts of a FileMaker Pro database. The various parts of FileMaker Pro integrate well, but a database is closed to the outside world of the OS and/or other applications. What if your calculation requires some information from another SDK? What if you wanted to create your solution to present to the user a very specific style of dialog boxes?

With AppleScript and the right scripting addition, you could achieve some sort of integration with another application. With some layout tricks, you could simulate dialogs. However, this just isn't the same as having a programmable feature within the application that would give you direct access to the OS or another SDK.

FileMaker Pro "External Function Plug-ins" can bridge this gap between the application's predefined interface and your need for a custom one. An API/SDK was released for FileMaker Pro 4.0 that allows you to create your own compiled code that can be referenced by a formula in a calculation, as an "External" function. When an External function is referred to, FileMaker Pro passes program control over to the external code you created. Your plug-in can then do what it needs to do and pass the needed information (and program control) back to FileMaker Pro.

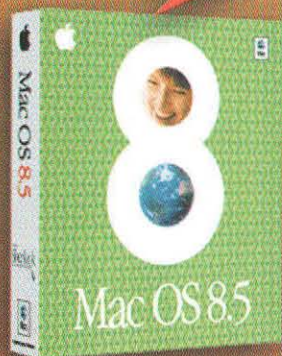
By using an External Function Plug-in, you can use calculation formulas that take advantage of things like recursion and looping. Plug-ins can hook into other applications or the Mac OS toolbox. This means things previously impossible with calculations and scripts are now within the database developer's reach.

For brevity and focus, this article assumes you have at least a rudimentary knowledge of FileMaker Pro calculation fields and scripts.

David McKee is the author of many FileMaker Pro plug-ins, including the "Full Example" plug-in for the FileMaker Pro Developer Edition. He presented a seminar on writing plug-ins at the 1998 FileMaker Developer Conference and works in the Engineering department of FileMaker, Inc.

These products and hundreds more!

Visit us at



Mac OS 8.5 **ONLY \$84.95!**

by Apple Computer, Inc.

Upgrade to the most powerful version of the Macintosh OS yet! Mac OS 8.5 offers faster file transfer, easier internet connectivity, and the powerfull search capabilities of Sherlock!

SPECIAL LIMITED TIME ONLY PRICE! (only while current supplies last)

PowerKey Rebound! **ONLY \$89!**

by Sophisticated Circuits

Rebound! is an ADB crash recovery system for the Macintosh. Its software monitors the system for crashes and restarts it automatically by sending a code from Rebound!



Resorcerer 2.2 **ONLY \$256!**

by Mathemaesthetics

The premiere Macintosh resource editor. Edits over 3 times as many resource types as ResEdit — including Alias resources and a new filtered TMPL for Mac OS 8.5 'icons' (Icon Suite) resources!



ChemOffice
\$289

Scripter 2.0
\$189

WebTen
\$395

ToolsPlus Lite
\$99

FaceSpan 3.0
\$189

Open GL
\$379

Adobe PageMill 3.0
\$94

VOODOO
\$199

Developer DEPOT®

PO Box 5200 • Westlake Village, CA • 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)
Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • E-mail: orders@devdepot.com

www.devdepot.com

The products you need, with the prices
and service you deserve... guaranteed.

AppMaker **ONLY \$189!**

by Bowers Development

Create your applications user interface with drag
and drop, point and click ease!



Funnel Web **ONLY \$189!**

by Active Concepts

High performance web site analysis and profiling. Provides full data
visualisation of server performance and client usage. Features include:

- Supports any web server on the planet 2D/3D charting Custom filters and settings
- Multi platform, Multi log and Gzip archive analysis Apple Scriptable (Mac OS only)
- Fully automated scheduling

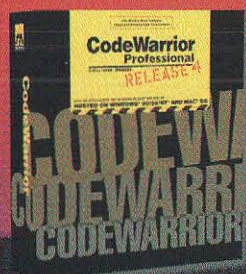
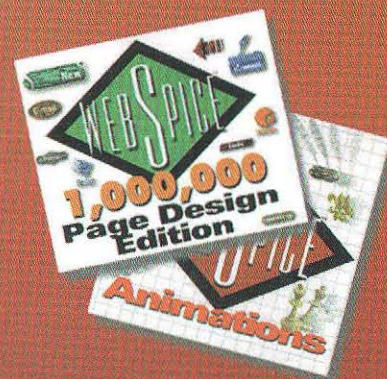


WebSpice Animation **ONLY \$89!**

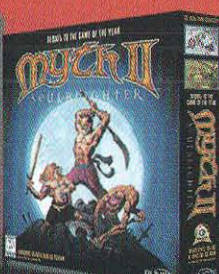
WebSpice 1,000,000 Page Design Edition **ONLY \$89!**

by DeMorgan Industries Corporation

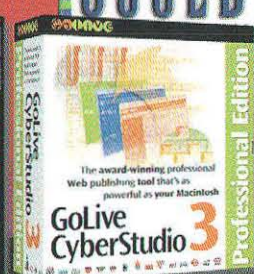
Create with royalty-free graphics and animation with 6 CD's of
high quality web graphics or 2 CD's of web animation!



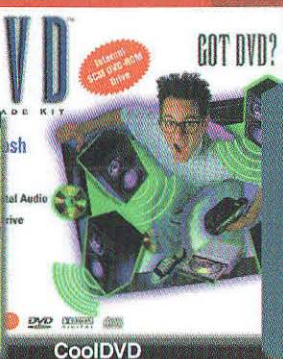
CodeWarrior Pro 4
\$349



Myth II Soulblighter
\$43.95



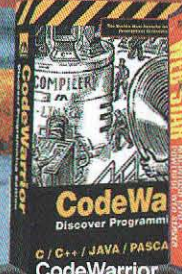
GoLive CyberStudio 3.1
\$279



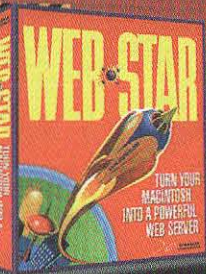
CoolDVD
\$449



Deadlock
\$46.95



Discover Programming
\$69



WebSTAR
\$419

TERMS

The following are terms used within this article. They are included for those new to programming plug-ins or modules.

- **Plug-in**

A plug-in is a file separate or external to the application that contains executable code. The application loads the plug-in at runtime and passes information and control to the plug-in whenever needed. The advantages of putting some features in a plug-in include easier revision of the feature, as well as allowing the application author to open up feature creation or modification to the public without exposing sensitive parts of the program.

A plug-in is essentially a separate application that is compiled as a code resource or library. It inhabits the same application space as the host program that loaded and passed execution to it.

- **Parameter Block**

A Parameter Block is the sole shared data structure that gets passed between the program and the plug-in. This allows data and functions to be passed seamlessly, even though they are basically two separate applications. If a particular function is not supported by the parameter block, then the desired control or communication between the two entities is probably not possible.

- **API**

An API, or Application Programming Interface, is a set of standards that your application must adhere to in order for it to be usable with the host program or standard. Some APIs are header and source code documents that you are required to use when compiling your code, while other APIs are just header files that allow you to link to a provided library.

There are no extra libraries needed to create a FileMaker Pro plug-in other than the ones you would normally need to create a code resource or shared library.

- **SDK**

An SDK, or Software Development Kit, is often used interchangeably with API.

- **External function**

An External function is a new class of calculation functions that can be used in a calculation formula. First introduced in FileMaker Pro 4.0, functions in this class are not built into the application. External functions can be designed and used in two ways.

When used from within a calculation field, as part of a formula, the External function behaves as a function. In this case, the External function returns data that will be used as part of the formula.

When used from a SetField script step, where an action is involved, the External function behaves as a command. In this case, what the External function returns may only be a result/error code.

- **External Function Plug-in**

An External Function Plug-in is a third-party file that contains External functions. These are loaded by the FileMaker Pro application upon startup. The External functions are then listed in the calculation dialog box interface of FileMaker Pro.

- **Feature String**

A Feature String is a series of flags that inform FileMaker Pro how to utilize the plug-in. Part of the Feature String is the "Feature String ID", which is essentially a Mac OS creator code that uniquely identifies that plug-in. No two plug-ins should have the same Feature String ID.

- **FileMaker Pro Developer Edition**

The FileMaker Pro Developer Edition (DE) is a FileMaker, Inc. product for high-end FileMaker Pro developers. This product includes many developer-oriented products that are of little use to end users. The Developer Edition (DE) is required to obtain the External Function Plug-in API. However, plug-ins created with the API can be used on any FileMaker Pro 4.0 (or greater) based product.

BEFORE STARTING

External Function Plug-ins (EFPs) were primarily intended to allow someone to supplement the list of calculation functions. While EFPs can supply many functions, you should consider a plug-in a toolbox of similar features. Try to focus your plug-in project on a certain function or tight group of functions. "Swiss army knife" plug-ins with unrelated functions will confuse end users and not distinguish your plug-in from others.

Non-programmers will use EFPs you create for their databases. Most FileMaker Pro database developers may not understand programming conventions that you take for granted. Design your plug-in and the syntax of the External functions with your audience in mind.

Since the Feature String ID is vital to your plug-in, you'll want to register it with Apple and FileMaker, Inc. Details of plug-in registration are included with the API package in the FileMaker Developer Edition.

If you have ever written a plug-in, a Desk Accessory, or a Driver, you will find that writing a FileMaker Pro plug-in is extremely simple. The details in this article concerning the API itself will probably be enough to get you on your way.

If you have never written a plug-in it is strongly recommended that you re-use the sample plug-in projects that come with the API. Walking a programmer through creating a FileMaker Pro plug-in from scratch is beyond the scope of this article.

THE EXTERNAL FUNCTION PLUG-IN API

The External Function Plug-in API package contains many files, and includes two examples. The best way to understand how the individual source and header files relate is to examine the examples. The core of the functionality of this API is actually quite simple.

The API includes a programming overview ("EFP Documentation.pdf") targeted at experienced programmers who also know FileMaker Pro. It outlines details relevant to the differences between a FileMaker Plug-in and other plug-ins. It covers only specifics that wouldn't be part of the creation of a "typical" plug-in. **It is not intended to be a "How-to" on writing plug-ins.**

The "FMFlags.h" header file contains used compiler directives to control code compilation. Since the API supports both Macintosh and Windows plug-ins on more than one compiler, this file allows you to have one set of source code files that will compile in all cases. **Do not alter this file.**

The "FMExtern.h" and "FMExtern.c" files define the parameter block and some shared function calls. The function calls are needed to, among other things, manipulate the "parameter" and "result" handles contained in the parameter block that your plug-in will use.

In the FMExtern.h file, you will notice the call-back functions for memory operations are defined. Below that, is the definition of the different kinds of plug-in events sent to the plug-in.

```
typedef enum { kFMXT_Init, kFMXT_Idle, kFMXT_Internal1,
               kFMXT_External, kFMXT_Shutdown,
               kFMXT_DoAppPreferences, kFMXT_Internal2 }
FMExternCallSwitch;
```

There will be an item in the parameter block that will be equal to one of these values. The "Internal" values are reserved items. When your plug-in receives control from the FileMaker Pro host program, it will be under one of the following conditions.

- **kFMXT_Init**

This indicates that your plug-in is being loaded during the initialization of the FileMaker Pro application. This occurs shortly after the user launches the application.

- **kFMXT_Idle**

This indicates that your plug-in is being given some processing time, but is not being explicitly used by the database or user. If the appropriate flag in the Feature String of the plug-in is not set, your plug-in will never receive this event.

- **kFMXT_External**

This indicates that one of the functions of your plug-in is being explicitly used by the database or user, and needs to return a value. If the appropriate flag in the Feature String of the plug-in is not set, the functions of your plug-in will not be accessible to the database or user.

- **kFMXT_DoAppPreferences**

This indicates that the user has just clicked on the "Configure" button in the plug-in management dialog within FileMaker Pro. Your plug-in can now present its own preferences dialog box. If the appropriate flag in the Feature String of the plug-in is not set, the Configure button will be grayed out and your plug-in will never receive this event.

- **kFMXT_Shutdown**

This indicates that the FileMaker Pro application is about to terminate normally. This allows your plug-in to do the appropriate cleanup, if needed.

Immediately below the FMExternCallSwitch definition is the type definition for "FMExternCallStruct". The FMExternCallStruct is the structure of the parameter block for

3D Graphics Software Development Kit

Everything you need to create stunning graphics on the Macintosh® with the industry standard API OpenGL®.

Create 3D images that use:

nurbs	gouraud shading	stenciling
lighting	anti-aliasing	texture mapping
fog	Z-Buffering	motion blurring
		and more!

Join the game writers, educators, interactive modelers, CAD designers, and scientists who are already using OpenGL.

Explore the possibilities of 3D programming.

- Complete with everything you need to learn OpenGL
- Mac OS, UNIX, MKLinux versions available
- Hardware acceleration

Demos available at our web site

 **Conix Graphics**

The Industry's Foundation for High-Performance Graphics

1.800.577.5505

www.conix3d.com

©Conix Enterprises, Inc. All rights reserved. OpenGL is a registered trademark of Silicon Graphics, Inc. Macintosh is a registered trademark of Apple Computer, Inc. All other trademarks are property of their respective owners.

the External function API. "FMExternCallPtr" is defined as a pointer to that struct. A global variable "gFMExternCallPtr" is then defined as a FMExternCallPtr.

These structures are the "meat and potatoes" of the communication between your plug-in and FileMaker Pro. Nested within this structure are two innocent looking variables of type long. They are "param3" and "result". The variable "param3" is actually the parameter data that was passed to the External function from the calculation formula in FileMaker Pro. It is referred to as "parameter" in some of the functions.

Variables you will coerce to "FHandle" variables in your plug-in code. A Fhandle is simply a handle to unsigned character data. There is no need for a length byte or null terminator for this string as the size of the handle is the size of the string. Since External functions can only return text/string data, the data you read from "param3" (or "parameter") and the data you put into "result" must be text, or a textual representation of a number.

The "FMTemplate.c" or "FMExample.c" files are examples of what the main file should look like. Note that they both define the main/entry point function for the various environments (PPC, 68k, or x86) a plug-in can be compiled for.

Inside the Main entry point, there is a switch statement that allows the plug-in to determine the condition under which control is passed to it, so that it may act appropriately.

```
switch (pb->whichCall) {
```

The variable "pb" is simply a local variable that was assigned to be equal to gFMExternCallPtr. Everything else within these files is non-essential to the API and specific to your plug-in.

Since the API is considered proprietary information, you cannot republish any information in any of the files other than the "FMTemplate.c" and "FMExample.c" files.

THE FULL EXAMPLE PLUG-IN

Provided with the API are a "Template" plug-in and a "Full Example" plug-in. The "Template" is a shell project with two functions typically used only by programmers.

The "Full Example" plug-in was designed as an example of what a "full blown" plug-in would look like. While it doesn't illustrate every possible use of a plug-in, it does have many subroutines that are well suited as External functions, or in designing them. It also demonstrates a rudimentary preferences dialog.

Let's say the FileMaker Pro user has installed the compiled Full Example plug-in and created a calculation field that refers to one of the functions, Xpl-BigPi. The reference to the External function would appear in the calculation formula as something like "...External("Xpl-BigPi","")...".

"External" signals FileMaker Pro that it must locate the function "Xpl-BigPi" in one of the loaded External Function Plug-ins. Once located, the External function passes the contents of the second parameter as a handle. In this case, the

handle will have a size of zero. The contents of this parameter are stored in the contents of "param3" for the plug-in.

Within the main/entry point, the plug-in will be notified that one of its External functions is being used, and that it must call the appropriate function. The sample plug-ins use a handler named "Do_External".

```
case kFMXT_External:
    Do_External(pb->parm2, (FHandle)(pb->parm3),
               (FHandle)(pb->result));
    break;
```

Note that the handler has three parameters (obtained from the parameter block). The first parameter is the index number of the plug-in. The first function is index number zero. Xpl-BigPi is the second function, so the index number will be one.

Going to the source code, the "Do_External" handler looks something like this:

```
// Received a message that some calc depends on one of the External functions in this
// plugin, so perform the appropriate function and return the results back to FileMaker
// Pro (FMP).
```

```
static void Do_External(long functionId, FHandle parameter,
                       FHandle result) {

    switch (functionId) {

        #if DEBUG_VERSION
        case 0:
            // Since we're debugging, let's have a debug function to access from FMP in
            // place of ThisVersion.
            // In FMP, we'll still have to "pretend" we're using the version function
            // "ThisVersion", but we're actually calling a different function in our
            // plugin.
            DebugPlugin(parameter, result); // not PluginVersion
            break;
        #else
        case 0:
            // This is the "shipping" version of our plug-in, so let's disable the debug
            // function and use the proper ThisVersion function -> PluginVersion
            PluginVersion(parameter, result);
            break;
        #endif

        case 1: // pi = Xpl-BigPi
            funct_PI(result);
            break;

        case 2: // format a number = Xpl-Format
            funct_Format(parameter, result);
            break;

        case 3: // number to words = Xpl-NumericalWords
            funct_Num2Words(parameter, result);
            break;

        default:
            // Bcep. This plugin is being sent a message to execute a function
            // that doesn't exist in this plug-in. (This could be due to a plug-in
            // installed with the same name as ours, with a bigger number of
            // functions.)
            #if FM_CPU_X86
            MessageBeep(0);
            #else
            SysBeep(1);
            #endif

    }; /* switch */
} /* Do_External */
```


The function with the index of one that is executed by this function is "funct_PI(result)". We pass on the result handle that this handler function received from the main/entry point. Remember that the parameter was "" in the original calculation formula.

The *pi* function doesn't need a parameter, so even if the handler passed on the parameter handle, it wouldn't use it. However, funct_PI does change the result handle to k_PI ("3!14159265358979\0"). It is defined as a null-terminated string in case the programmer will use it elsewhere. k_PILength represents the length of the string we will actually return to FileMaker Pro (i.e. 16).

The exclamation point is a placeholder that will be replaced with whatever decimal point character is appropriate for the country the plug-in is being used in.

```
// Returns pi to the 14th digit (past decimal), as hardcoded in the defines file
// "FMExample.h"
```

```
static void funct_PI(FHandle result) {
    FMX_SetHandleSize(result, k_PILength);

    if (FMX_MemoryError() == 0) {
        #if FM_CPU_X86
            lstrcpy((LPTSTR)*result, k_PI);
        #else
            BlockMoveData(k_PI, *result, k_PILength);
        #endif
    }

    (*result)[1] = decimalSepCH;
    // Make sure to substitute that 'I' in our pi value to the
    // acceptable decimal separator for numbers in this
    // country.
} // end if
} // funct_PI
```

When control is passed back to FileMaker Pro, the result handle in the parameter block is the value of "External("Xpl-BigPI","")".

Although the other functions do much more complicated things, this is essentially how all External functions will behave. The rest is up to your imagination!

CONSIDERATIONS

Debugging code resources or shared libraries is not a simple task. However, if you use DebugStr() and the "DisplayMessage" function, you can do rudimentary debugging very easily. DebugStr() is a function defined in the Universal Headers, which allows you to display a Pascal style string in a low-level debugger. "Display Message" is a function that is defined in the sample plug-in projects.

If you are having trouble, you might want to consider pasting your plug-in functions into a console project and debug it as an application.

The API is provided only in a C/C++ format. If you plan on using another programming language, you will have to convert the API from C to the other language.

The API is designed to be cross-platform (both Mac OS and Windows). If your plug-in is going to be Mac OS only, you can feel free to strip out all the compiler directives for readability. It is recommended to keep them in, however.

Many FileMaker Pro database solutions are cross-platform.

In addition to having cross-platform compiler directives, the Full Example plug-in takes into consideration international users. If you are manipulating numbers, dates or monetary amounts, it is strongly recommended that you also take international users into consideration. The Internet has significantly increased the market for products that are friendly to international users.

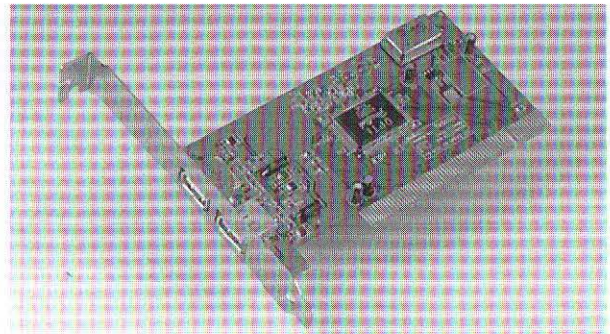
Some advanced plug-ins will rely on resources. Make sure to set and restore the current resource file to avoid getting a resource from another plug-in. Also, use resource IDs only in the range between 23,000 and 24,999.

While you own the code you create, you should be careful not to release or publish any files that include code from the External Function Plug-in API. You should also treat your plug-in as a separate software product with its own user license agreement and provisions for technical support. FileMaker, Inc. does not warrant the usability of third-party plug-ins, nor provide technical support for using them.

MT

USB-Stuff

Hubs, cards, mice, keyboards, joy sticks, cameras, adaptors, modems, monitors, speakers...
more USB stuff than anyone!



2 Port USB to PCI interface card for the Mac by

ADS Technologies

WWW.USBSTUFF.COM

1-888-USB-USB-US

707-778-6299

by Bob Boonstra, Westford, MA

SPHERE PACKING

This month we're going to help you recover from the clutter that might result from the holiday season. Imagine that your post-holiday household is filled with gifts, all of which have to be put somewhere. Imagine further that those gifts include sports equipment given to your children, or parents, or siblings, or grandchildren, as the case may be. And finally, imagine that the sports equipment includes a collection of balls of various sizes – basketballs, baseballs, soccer balls, beach balls, etc. (OK, if I've stretched your imagination to the breaking point, think of some other reason you might have a large collection of spherical objects.) We've got to find somewhere to store all of those balls, and space is at a premium. Fortunately, we also have a very large collection of boxes of various sizes, so many, in fact, that you can count on finding a box of the exact size that you might need. In keeping with our desire for a few less difficult problems, your Challenge is to pack the balls into the smallest box possible, so that we can store them efficiently.

The prototype for the code you should write is:

```
#if defined(__cplusplus)
#if defined (__cplusplus)
extern "C" {
#endif

typedef struct Position {
    double coordinate[3]; /* coordinate[0]==X position, [1]==Y, [2]==Z */
} Position;

void PackSpheres(
    long numSpheres, /* input: number of spheres to pack */
    double radius[], /* input: radius of each of numSpheres spheres */
    Position location[] /* output: location of center of each sphere */
);

#if defined (__cplusplus)
}
#endif
```

Your `PackSpheres` routine will be given the number of balls (`numSpheres`) to be packed away, along with the `radius` of each of those spheres. The task is simple. Arrange the collection of balls into a rectangular parallelepiped ("box") such that no ball intersects any other ball (i.e., the distance between the centers of any two balls is greater than or equal to the sum of their radii). `PackSpheres` returns back the coordinates of the center of each ball in the `location` parameter. Your objective is to minimize the volume of the box that contains all the balls, where the extent of the box in each dimension (X, Y, and Z) is determined by the maximum and minimum coordinates of the balls, considering both the `location` of the center of the ball and its `radius`.

While you must ensure that the balls do not intersect, you need not ensure that the balls touch. In our storage room, boxes of balls can contain balls that levitate in the open space between other balls.

The winner will be the solution that minimizes the volume of the box containing all the balls, plus a penalty of 1% of additional storage volume for each millisecond of execution time.

This will be a native PowerPC Challenge, using the latest CodeWarrior environment. Solutions may be coded in C, C++, or Pascal.

THREE MONTHS AGO WINNER

Congratulations to **Pat Brown (Staunton, VA)** for submitting the winning solution to the October Hearts Challenge. Pat's solution beat the second-place entry submitted by Tom Saxton and "dummy" entries that rounded out a tournament of four players. Pat's solution was both the faster of the two and the more successful at avoiding point cards, capturing approximately one third fewer points than Tom's solution.

Pat's strategy was fairly simple. His passing strategy is to pass the three highest cards in his hand. By not including a low heart in the pass, this strategy can aid a shoot attempt by an opponent, as well as being dangerous if it passes the queen of spades to the left. When leading, the playing strategy is to force out the queen of spades as quickly as possible, unless of course he has the queen. While he does not attempt to "shoot the moon", he is watchful for attempts by other players to shoot, and holds on to high cards until any potential shoot is spoiled. Otherwise, Pat tries to play the highest legal card that is lower than the current trick leader.

Tom submitted two solutions, a simple one (used in the tournament at Tom's request), and a more sophisticated (but less successful) one. The simple solution also tries to avoid taking tricks and does not attempt to shoot. It is a little more clever in selecting the pass, in that it tries to create a void if possible. It does not keep track of who might be attempting to shoot, and therefore does not attempt to stop them. Tom's second player keeps track of who is void in what suits and tries to shoot when it has a strong hand. However, it isn't quite perfected, and does much worse in a tournament than the first player.

I've included a Point Comparison chart that helps explain the performance of the two players. The vertical bars indicate the number of hands in which each player captured the number of points shown along the horizontal axis. You can see that Pat's solution was slightly more successful at capturing fewer than 4 points in a hand, very successful at avoiding being stuck with the queen of spaces, and extremely effective at capturing fewer than 20 points in a hand. The line graphs show the cumulative effect of the respective strategies on the score.

The table below summarizes the scoring for Pat and Tom's Hearts entries. The teams played a total of 24 matches, consisting of

over 25000 hands of 13 tricks each. The Total Points column in the table lists the number of hearts captured during all of those tricks, plus 13 points for each Queen of Spades, and -26 points for each shoot. The table shows the number of tricks "won" by each player, and the number of times each successfully "shot the moon". You can see that Pat's winning solution did not attempt to shoot, and was very successful at avoiding being stuck with all of the point cards. Although not shown in the table, the less-than-intelligent "dummy" players "shot the moon" more often than either Pat or Tom. This was a consequence of their simplistic "strategy" for not taking points, which led them to hold on to high cards longer than a more sophisticated player would have done. Also shown in the table are the execution time of each solution in milliseconds, the total score, including the penalty of one point per millisecond of execution time, and the code and data sizes. As usual, the number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges prior to this one.

Name	Total	Tricks	Shoots	Time	Score	Code
Data	Points	Won		(msec)	Size	Size
Pat Brown		94775	59703	1	833	95608
3152				398		
Tom Saxton (49)	157105	80902	67	1230	158335	
2548			72			

TOP CONTESTANTS

Listed here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated 20 or more points during the past two years. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	204	10.	Nicollé, Ludovic	34
2.	Saxton, Tom	59	11.	Lewis, Peter	31
3.	Boring, Randy	56	12.	Hart, Alan	21
4.	Mallett, Jeff	50	13.	Antoniewicz, Andy	20
5.	Rieken, Willeke	47	14.	Brown, Pat	20
6.	Cooper, Greg	44	15.	Day, Mark	20
7.	Maurer, Sebastian	40	16.	Higgins, Charles	20
8.	Heithcock, JG	37	17.	Hostetter, Mat	20
9.	Murphy, ACC	34	18.	Studer, Thomas	20

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	2 points
suggesting Challenge	2 points

Here is Pat's winning Hearts solution:

MyHearts.c

Copyright © 1998 Pat Brown

```
#include "Hearts.h"

/* *****
 * Hearts.c
 *
 * Author: Pat Brown
 *
 * Trivia: there are 5.36447377655x10^28 different ways that a deck can be dealt
 * out for a game of Hearts.
 */

#define gMAX 52
#define gMIN 1

// Just the relative rankings, 1..52.
// Can be referenced easily using spot and suit value.
/*
    2C,2D,2S,3C,3D,3S,4C,4D,4S,5C,5D,5S,6C,6D,6S,7H,7H,7H,7H,
    5H,6H,7C,7D,7S,8C,8D,8S,9C,9D,9S,7H,8H,9H,10C,10D,10S,
    JC,JD,JS,QC,QD,KC,KD,AC,AD,10H,JH,QH,KS,AS,KH,AH,QS
 */
static const int gCardValue[5][14] =
{
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // NoSuit
    {0, 3, 6, 9, 12, 15, 22, 25, 28, 35, 38, 52, 48, 49}, // Spade
    {0, 16, 17, 18, 19, 29, 30, 31, 32, 45, 46, 47, 50, 51}, // Heart
    {0, 2, 5, 8, 11, 14, 21, 24, 27, 34, 37, 40, 42, 44}, // Diamond
    {0, 1, 4, 7, 10, 13, 20, 23, 26, 33, 36, 39, 41, 43} // Club
};

inline int getCardValue(theSuit, theSpot)
{ return gCardValue[theSuit][theSpot]; }

inline UInt16 NEXTSEAT(UInt16 theSeat)
{ return (theSeat+1)%4; }

/* *****
 * Prototypes
 * *****
 */
inline static SInt16 findThisCard(const Card theCards[13],
    const Suit theSuit, const Spot theSpot);
static UInt16 findHighestLimitCard(const Card theCards[13],
    const int uLimit);
static UInt16 findLowestLimitCard(const Card theCards[13],
    const int lLimit);
static UInt16 findHighestIndexLimitCard(const Card
    theCards[13],
    const UInt16 valid[13], const int numValid, const int
    uLimit);
static UInt16 findLowestIndexLimitCard(const Card
    theCards[13],
    const UInt16 valid[13], const int numValid, const int
    lLimit);
static int findValidCards(const Card theCards[13],
    const Suit theSuit, UInt16 validCards[13]);

inline UInt16 findHighestCard(const Card theCards[13])
{ return findHighestLimitCard(theCards, gMAX); }
inline UInt16 findLowestCard(const Card theCards[13])
{ return findLowestLimitCard(theCards, gMIN); }
inline UInt16 findHighestIndexCard(const Card theCards[13],
    const UInt16 valid[13], const int numValid)
{ return
    findHighestIndexLimitCard(theCards, valid, numValid, gMAX); }
inline UInt16 findLowestIndexCard(const Card theCards[13],
    const UInt16 valid[13], const int numValid)
{
    return
    findLowestIndexLimitCard(theCards, valid, numValid, gMIN);
}
```



```

/*****

```

Global variables

```

*****/
static UInt16 mySeat;
// hasPoints is a bitfield of which players have
// taken points in this hand
static int hasPoints;
// another bitfield of what suits I'm cutting
static int cuttingSuit;
static int tryToSpoil;
/*****
runningTotal keeps track of "points" for each player
in a hand. These aren't real points, but a key to
watch for someone trying to pull a sweep.
If someone gets more than six hearts, or the queen of
spades and three hearts, they may be trying to sweep
(unless someone else has points).
*****/
static int runningTotal[4];
static Boolean blackLadyInHand;
static Boolean blackLadyPlayed;
static Boolean heartsBroken;

```

Required functions

```

*****/

InitTournament
pascal void InitTournament(const UInt16 numPlayers,
                           const UInt16 gameEndingScore)
{
    return;
}

InitGame
pascal void InitGame(const UInt32 playerID[4],
                    const UInt16 yourSeat)
{
    mySeat = yourSeat;
}

SelectPass
pascal void SelectPass(const Card dealtHand[13],
                      const Pass passDirection, UInt16 passedCards[3])
{
    int i, top, newtop;

// Init globals at the start of a hand.
    blackLadyPlayed = heartsBroken =
        tryToSpoil = hasPoints = cuttingSuit =

runningTotal[0]=runningTotal[1]=runningTotal[2]=runningTotal[3]
    = 0;

    if (passDirection != kNoPass)
    {
        top = gMAX;
        for (i=0; i<3; i++) // get the three highest cards
        {
            newtop = passedCards[i] =
                findHighestLimitCard(dealtHand, top);
            top = getCardValue(dealtHand[newtop].suit,
                             dealtHand[newtop].spot)-1;
        }
    }
}

PlayTrick
pascal void PlayTrick(const UInt16 trickNumber,
                     const UInt16 trickLeader, const Card yourHand[13],
                     const Card cardsPlayed[4], UInt16 *yourPlay)
{
    UInt16 validCards[13], myCard;
    int num, i;
    Suit leadingSuit;
    Spot whatsTaking;
    UInt16 whosTaking;
    Boolean pointsTaking;

```

```

    if (trickNumber == 0)
    {
        // see if I've been passed the Queen of Spades
        blackLadyInHand =
            (findThisCard(yourHand, kSpade, kQueen) >= 0);
        if (trickLeader == mySeat)
        {
            *yourPlay = findThisCard(yourHand, kClub, k2);
            return;
        }
        else
            goto NOTLEADING;
    }

    if (trickLeader == mySeat)
    {
        // try to force the Queen of Spades (unless I have it)
        if (!blackLadyPlayed && !blackLadyInHand)
        {
            if (
                (num = findValidCards(yourHand, kSpade, validCards)) != 0)
            {
                myCard = findLowestIndexCard(yourHand, validCards, num);
                // don't play higher than the Queen
                if (yourHand[myCard].spot < kKing)
                {
                    *yourPlay = myCard;
                    return;
                }
            }
            // (!blackLadyPlayed && !blackLadyInHand)
            num = findValidCards(yourHand, kNoSuit, validCards);
            *yourPlay = findLowestIndexCard(yourHand, validCards, num);
            return;
        }
        // if (trickLeader == mySeat)

NOTLEADING:
        leadingSuit = cardsPlayed[trickLeader].suit;
        num=0;
        // this is faster than scanning yourHand every time
        if ((cuttingSuit & (1 << leadingSuit)))
            goto CUTTING;
        if (
            (num = findValidCards(yourHand, leadingSuit, validCards))=1)
        {
            // only one card we can play
            *yourPlay = validCards[0];
            return;
        }
        if (num == 0)
        {
            cuttingSuit |= 1 << leadingSuit;
            goto CUTTING;
        }
        if (num == 0)
        {
CUTTING:
            // we're cutting this suit
            if (trickNumber == 0)
            {
                // Can't play points on the first trick.
                // 51 keeps us from playing the Queen of Spades.
                num = findValidCards(yourHand, kNoSuit, validCards);
                *yourPlay =
                    findHighestIndexLimitCard(yourHand, validCards, num, 51);
                return;
            }
            if (!tryToSpoil)
                *yourPlay = findHighestCard(yourHand);
            else // Save the high cards to spoil a sweep.
                *yourPlay = findLowestCard(yourHand);
            return;
        }
        // if (num == 0)

        // See who's winning this trick so far.
        i = trickLeader;
        pointsTaking = leadingSuit == kHeart;
        whatsTaking = kNoSpot;
        while (i != mySeat)

```



```

{
    if ( (cardsPlayed[i].suit == kSpade) &&
        (cardsPlayed[i].spot == kQueen))
        pointsTaking = true;
    if (cardsPlayed[i].suit == leadingSuit)
    {
        if (cardsPlayed[i].spot > whatsTaking)
        {
            whatsTaking = cardsPlayed[i].spot;
            whosTaking = i;
        }
    }
    else // (cardsPlayed[i].suit != leadingSuit)
    {
        pointsTaking |= (cardsPlayed[i].suit == kHeart);
    } // if (cardsPlayed[i].suit == leadingSuit) else
    i = NEXTSEAT(i);
} // while (i != mySeat)

if ((leadingSuit == kSpade) && blackLadyInHand)
{
    myCard = findThisCard(yourHand, kSpade, kQueen);
    if (whatsTaking > kQueen) // dump it on King or Ace
    {
        *yourPlay = myCard;
        return;
    }
    else // don't play the Queen of Spades
    {
        for (i=0; i<num; i++)
        {
            if (validCards[i] == myCard)
            {
                while (++i < num)
                    validCards[i-1] = validCards[i];
                num--;
            }
        } // for (i=0; i<num; i++)
    } // if (whatsTaking > kQueen) else
} // if ((leadingSuit == kSpade) && blackLadyInHand)

if (trickLeader == NEXTSEAT(i)) // playing last
{
    if (!pointsTaking)
    {
        if (tryToSpoil) // don't waste the high cards
        {
            *yourPlay =
                findLowestIndexCard(yourHand, validCards, num);
        }
        else
        {
            *yourPlay =
                findHighestIndexCard(yourHand, validCards, num);
        }
        return;
    }
    else
    {
        if (tryToSpoil == (1 << whosTaking))
        {
            myCard = findHighestIndexCard(yourHand, validCards, num);
            if (yourHand[myCard].spot < whatsTaking) // we can't take this trick
            {
                myCard = findLowestIndexCard(yourHand, validCards, num);
                *yourPlay = myCard;
                return;
            }
        }
        else // someone else is spoiling the sweep
        {
            *yourPlay =
                findHighestIndexLimitCard(yourHand, validCards, num,
                    getCardValue(leadingSuit, whatsTaking));
            return;
        }
    } // if (!pointsTaking) else
} // if (trickLeader == NEXTSEAT(i))
if (tryToSpoil)
{
    myCard = findHighestIndexCard(yourHand, validCards, num);
    if (yourHand[myCard].spot < whatsTaking) // we can't take this trick
    {
        myCard = findLowestIndexCard(yourHand, validCards, num);
        *yourPlay = myCard;
        return;
    }
}

// Standard behaviour
// Play the highest card under the current highest card.

```

```

*yourPlay = findHighestIndexLimitCard(yourHand, validCards,
    num, getCardValue(leadingSuit, whatsTaking));
return;
}

```

TrickResults

```

pascal void TrickResults(const Card lastTrick[4],
    const UInt16 trickWinner)
{
    // Keep track on who has what points so we can watch for a sweep.
    int i;
    int points = 0;
    int whoWon;

    for (i=0; i<4; i++)
    {
        switch (lastTrick[i].suit)
        {
            case kSpade:
                if (lastTrick[i].spot == kQueen)
                {
                    // not 13, we trigger a sweep when "points" hit 7
                    points += 4;
                    blackLadyPlayed = true;
                    blackLadyInHand = false;
                }
                break;
            case kHeart:
                points++;
                heartsBroken = true;
                break;
        }
    } // for (i=0; i<4; i++)
    if (points == 0)
        return;
    points = runningTotal[trickWinner] += points;
    whoWon = 1 << trickWinner;
    hasPoints |= whoWon;
    if (tryToSpoil)
    {
        if (whoWon != tryToSpoil)
            tryToSpoil = 0;
    }
    else // (!tryToSpoil)
    {
        if (trickWinner != mySeat)
            tryToSpoil = ((hasPoints == whoWon) && (points > 6))
                ? whoWon
                : 0;
    }
}

```

HandResults

```

pascal void HandResults(const SInt16 pointsThisHand[4],
    const SInt32 cumPoints[4])
{
    return;
}

```

```

/*****
My functions
*****/

```

findThisCard

```

/*****
Return the index of a particular card,
or -1 if it's not there.
*****/
inline static SInt16 findThisCard(const Card theCards[13],
    const Suit theSuit, const Spot theSpot)
{
    SInt16 c = -1, i = 13;

    while ((c<0) && i--)
    {
        if ((theCards[i].spot == theSpot) &&
            (theCards[i].suit == theSuit))
            c = i;
    }
    return c;
}

```


findHighestLimitCard

These next few functions find cards based on a limit.
(it's an inclusive limit)
If there are no cards within the limit, then call
the opposite function.

```
static UInt16 findHighestLimitCard(const Card theCards[13],
    const int uLimit)
```

```
{
    int i = 13;
    int points, theValue = 0;
    SInt16 theCard = -1;

    while (i-)
    {
        points = getCardValue(theCards[i].suit, theCards[i].spot);
        if ((points > theValue) && (points <= uLimit))
        {
            theCard = i;
            if (uLimit == (theValue = points))
                return theCard;
        }
    }
    if (theCard < 0)
        theCard = findLowestLimitCard(theCards, uLimit);
    return theCard;
}
```

findLowestLimitCard

```
static UInt16 findLowestLimitCard(const Card theCards[13],
    const int lLimit)
```

```
{
    int i = 13;
    int points, theValue = 99;
    SInt16 theCard = -1;

    while (i-)
    {
        points = getCardValue(theCards[i].suit, theCards[i].spot);
        if ((points < theValue) && (points >= lLimit))
        {
            theCard = i;
            if (lLimit == (theValue = points))
                return theCard;
        }
    }
    if (theCard < 0)
        theCard = findHighestLimitCard(theCards, lLimit);
    return theCard;
}
```

findHighestIndexLimitCard

Index functions use an array of valid card indexes.

***** INFINITE RECURSION WARNING *****

Do not call these functions with numValid == 0!!!

```
static UInt16 findHighestIndexLimitCard(const Card
theCards[13],
    const UInt16 valid[13], const int numValid,
    const int uLimit)
```

```
{
    int i = numValid;
    int points, theValue = 0;
    SInt16 theCard = -1;

    while (i-)
    {
        points = getCardValue(theCards[valid[i]].suit,
            theCards[valid[i]].spot);
        if ((points > theValue) && (points <= uLimit))
        {
            theCard = valid[i];
            if (uLimit == (theValue = points))
                return theCard;
        }
    }
}
```

```
if (theCard < 0)
    theCard = findLowestIndexLimitCard(theCards, valid, numValid,
        uLimit);
return theCard;
}
```

findLowestIndexLimitCard

```
static UInt16 findLowestIndexLimitCard(const Card theCards[13],
    const UInt16 valid[13], const int numValid,
    const int lLimit)
```

```
{
    int i = numValid;
    int points, theValue = 99;
    SInt16 theCard = -1;

    while (i-)
    {
        points = getCardValue(theCards[valid[i]].suit,
            theCards[valid[i]].spot);
        if ((points < theValue) && (points >= lLimit))
        {
            theCard = valid[i];
            if (lLimit == (theValue = points))
                return theCard;
        }
    }
    if (theCard < 0)
        theCard = findHighestIndexLimitCard(theCards, valid, numValid,
            lLimit);
    return theCard;
}
```

findValidCards

Fill an array with indexes to cards of a particular suit.
Returns the number of valid cards (the size of the array)
Passing kNoSuit will fill the list depending on whether
hearts can be played now.

```
static int findValidCards(const Card theCards[13],
    const Suit theSuit, UInt16 validCards[13])
```

```
{
    UInt16 i;
    int num = 0;

    if (theSuit == kNoSuit)
    {
        if (heartsBroken)
        {
            for (i=0; i<13; i++)
                if (theCards[i].suit != kNoSuit) validCards[num++] = i;
        }
        else // (!heartsBroken)
        {
            for (i=0; i<13; i++)
            {
                if ((theCards[i].suit != kHeart) &&
                    (theCards[i].suit != kNoSuit))
                    validCards[num++] = i;
            }
            if (num == 0) // Nothing but hearts left to play.
            {
                for (i=0; i<13; i++)
                    if (theCards[i].suit != kNoSuit) validCards[num++] = i;
            }
        } // if (heartsBroken) else
    }
    else // (theSuit != kNoSuit)
    {
        for (i=0; i<13; i++)
        {
            if (theCards[i].suit == theSuit)
                validCards[num++] = i;
        }
    }
    return num;
}
```

147

by Darryl Lovato, Brian Pfister, Peter Thomas, and Dave Mark, ©1998 by Metrowerks, Inc., all rights reserved.

What's New at Aladdin?

This month, the Factory Floor visits with the folks at Aladdin. My personal connection to Aladdin began when they first introduced Magic Menu, the iconic menu that was added to the Finder's menu bar and allowed you to perform a variety of compression operations on the current Finder selection. At the time, I was blown away by this trick and trying to figure out how they did that opened up a bunch of new Mac programming doors for me.

Darryl Lovato is Aladdin VP, Chief Technology Officer, and Board member. Lovato, 32, is a 15 year veteran in the high tech software business and an Aladdin co-founder. Prior to co-founding Aladdin, Lovato worked as a Software Engineer for Dayna Communications on "Mac Charlie", a PC-Mac Emulator, and FT100, a DOS drive for the Macintosh, while attending the University of Utah in the Computer Science Department. During this time Darryl also worked as a consultant, a contract programmer, and a contributing editor of MacTech Magazine.

Darryl also worked at TML Systems, developing the TML Pascal Compiler, as well as at Apple Computer, where he worked on the Apple IIGS Finder, Control Panels, Advanced Disk Utilities products and later worked on the Macintosh IIsi and Macintosh IIsx computers. Lovato also served as the Apple II liaison to the Apple human interface group.

Brian Pfister is the StuffIt Deluxe Lead Engineer. Pfister, 32, an avid soccer player and computer engineer, attended the University of Santa Clara, majoring in Computer Science. Afterwards, Pfister moved to South West Florida to start his company, Softlink, where he developed OpenDoc parts.

Pfister's expertise led him to Aladdin Systems to continue his engineering focus. Over the past year, Pfister has enjoyed working on the Macintosh-standard compression software, StuffIt Deluxe, which many engineering aspects in version 5.0 are accredited to.

Peter Thomas, 29, has worn many hats for the three and a half years he's been with Aladdin. He began his career in technical support, but has also held positions in tradeshow technical management, product demonstrations, and is currently, a product manager for Aladdin FlashBack, Private File and Aladdin Desktop Magician.

Dave: How can developers add Stuffit technology to their own apps?

Darryl: Aladdin provides a StuffIt Engine SDK (software developer kit), that is downloadable from the Aladdin website <<http://www.aladdinsys.com/developers/>>. The original purpose of the engine was to put all the code that does compression and decompression in one place, so it wouldn't be duplicated in all of Aladdin's products. A "side-effect" of this made it possible for other developers to call the engine to compress or decompress files. Many developers have already included this functionality in their products including America Online and QUALCOMM.

Using the StuffIt Engine is actually pretty easy. You simply open the engine, pass the files or folders to compress or decompress to the engine, and it does the rest. I've included some sample code that shows how simple it really is. The StuffIt Engine SDK also provides low-level calls that allow you to do things like walk the contents of an archive, etc.

/* SimpleTest.c puts up a dialog box to allow you to select a file. It then determines the file type and takes appropriate action of stuffing (if the selected file is not a .sit file) or unstuffing (if the selected file is a .sit archive). Modifying constants in the code allows you to change the stuffing parameters and see their effects. SimpleTest.c demonstrates how to use the high level SDK calls.*/

```
#ifndef _MWERKS_
#include <Desk.h>
#include <Dialogs.h>
#include <DiskInit.h>
#include <Errors.h>
#include <Events.h>
#include <Files.h>
#include <Fonts.h>
#include <Memory.h>
#include <Menus.h>
#include <OSEvents.h>
#include <StandardFile.h>
#include <TextEdit.h>
#include <Types.h>
#include <Windows.h>
#endif
#include "StuffItEngineLib.h"
```



```

main()
{
    OSErr      err;
    StandardFileReply theReply;
    FSSpec      theFSSpec, destSpec, expandedSpec;
    long        magicCookie;
    Boolean     makeSEA = true;
    // changes these for different actions...
    Boolean     encodeBinHex = true;

    InitGraf((Ptr)&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(0L);
    InitCursor();

    StandardGetFile(NULL, -1, NULL, &theReply);
    if (theReply.sfGood) {
        theFSSpec = theReply.sfFile;
        err = OpenSITEngine(kUseExternalEngine, &magicCookie);
        // first we open the engine up
        if (!err && magicCookie) {
            // if we found it - let's use it!
            // make sure we have the correct version of the Engine
            if (GetSITEngineVersion(magicCookie) >=
                kFirstSupportedEngine) {
                short ftype = DetermineFileType(&theFSSpec);

                if (ftype == ftype_StuffIt) { // then unstuff it
                    UnStuffFSSpec(magicCookie, &theFSSpec, NULL, NULL,
                        kDontPromptForDestination, kDontDeleteOriginal,
                        createFolder_Smart, kSaveComments, kDontStopOnAbort,
                        USIT_conflict_autoRename, textConvert_Smart);
                } else { // stuff that sucker!
                    FSSpec      arcSpec;
                    FSSpecArrayHandle specListH = NewFSSpecList();

                    if (ftype) { // this is something we can expand
                        ExpandFSSpec(magicCookie, ftype, &theFSSpec, NULL, NULL,
                                    createFolder_Smart, kDontDeleteOriginal,
                                    textConvert_Smart);
                    } else { // stuff it!
                        AddToFSSpecList(&theFSSpec, specListH);

                        GetNewArchiveFSSpec(specListH, makeSEA, &arcSpec);
                        StuffFSSList(magicCookie, specListH, &arcSpec, NULL,
                                    kCompressOriginal, kDontDeleteOriginal,
                                    kDontEncryptOriginal, kResolveAliases,
                                    kRecompressCompressed, SIT_conflict_autoRename);

                        if (makeSEA)
                            ConvertSIT2SEA(magicCookie, &arcSpec);

                        if (encodeBinHex) {
                            HQXEncodeFSSpec(magicCookie, &arcSpec, NULL,
                                kDontDeleteOriginal, kDontAddLineFeeds,
                                kDefaultHQXCreator);
                        }
                    }

                    DisposeFSSpecList(specListH);
                }
            }

            CloseSITEngine(magicCookie); // and finally close it
        }
    }
}

```

Dave: By the time folks read this, StuffIt 5.0 will have hit the streets. What new technologies are in this release?

Darryl: The biggest change in StuffIt 5.0 is that for the last three years we have been re-writing the STUFFIT archive file format from the ground up. The old format

was about six years old, and didn't support a lot of new technology that has been created since then. The new format has the following new features:

- New Stuffit technology provides a 20% compression improvement and gives two compression methods, "Fast" or "Maximum".

Fast is roughly equivalent to the old Stuffit algorithm, except that it's about 20% faster

Maximum is brand new, and compresses files an average of 20% smaller than the old algorithm

- Multi-Lingual - All strings are stored in Unicode, and file names can now be of any length
- Updated transparent cross-platform file format interchange, so it is now more multi-platform. All OS file coding information is saved/translated/restored correctly, regardless of source or target platform, (Mac, Windows, Unix). A type/creator extension table also maps file types between platforms.

Stuffit Deluxe now includes full support of HFS+ on the Mac. The create and modification dates are all translated correctly and Mac file comments are now saved and restored.

- No more archive file size or number limits
- In addition to the new file format, the new version now supports MacBinary III, OS 8.5, Appearance Manager, Archive Via Rename support for .hqx and .bin, and support for MS Outlook Express and Mailsmith Support has been added to the Mail extension.

Dave: Can you fill us in on the magic behind your True Finder Integration technology?

Brian: True Finder Integration (TFI) is a series of components that allow you to manipulate Stuffit archives from within the Finder. The TFI Control Panel is the heart of these components. TFI has a plugin architectural.

```

_mpeg.extension", "mpg.mpeg");
user_pref("mime.audiCurrently we have 3 plugins
(MagicMenu!", ArchiveViaRename!", and StuffIt
Browser"!);

```

We load the plugins in a 2 step process. TFI does its preflight at INIT time. That includes installing the appropriate Gestalt Vectors that the individual plugins will use to communicate with TFI. TFI then finds all the

plugins and reads their individual internal preferences. One of these preferences tells TFI when the plug-in should be called for its pre-flighting. This can be either immediately, when Finder launches, or at both times. If there is a plug-in that requires a preflight at Finder launch, TFI will patch InitDialogs. From within this patch TFI will call the plugin to initialize itself.

Each plugin follows a similar loading procedure. They first install a Gestalt vector so that the TFI control panel can configure them as needed. They then load the necessary code resources and reserve the needed memory. Lastly they patch the calls that they need. AVR is the simplest, patching only _Rename. The Browser is the most complicated. It patches more than 40 OS traps & ToolBox traps. MagicMenu is pretty modest, patching only the four MenuBar related system calls: _DrawMenuBar, _MenuKey, _MenuDispatch and _MenuSelect. We use a head patch with DrawMenuBar. We head and tail patch the other three.

Like TFI, MagicMenu has a plugin technology. Its main purpose is to determine what is selected in the Finder and pass that information to the appropriate MagicMenu extension (MME). It is the specific MME that makes the appropriate calls to the StuffIt Engine. MagicMenu and its plugins communicate through a dispatch routine.

Dave: What are your plans for Carbon/Mac OS X?

Brian: It is still too early to commit to Carbon/Mac OS X 100%. We are keeping a very close eye on it for future revisions of our products. We expect a future version of Deluxe to be very close to 100% carbon compliant but since Carbon/Mac OS X development will be progressing in time frame parallel to our engineering and development we can not expect Deluxe to be completely compliant.

The True Finder integration technologies are a completely different challenge, since they are so closely tied to the current OS architecture. We have conceded that they will require a complete rewrite to work under Mac OS X. Given that fact, we intend to look closely at Mac OS X and see how we can best fit the functionality of TFI into the Mac OS X. This will be better understood as Mac OS X evolves over the next year.

Dave: What can you tell me about your latest release of InstallerMaker, V5.3?

Brian: InstallerMaker has gone through dramatic changes in the last year. We began at MacWorld Expo '98, by introducing support for retrieving files from the Internet. We also gave installers a face-lift, to provide

a more modern look and feel. In our 5.0 release, following Apple's Worldwide Developer's Conference, we radically altered the internal structure and handling of archives and installers. For the first time, developers could set all conditions, package assignments, and actions individually to items nested inside folders. We also added long overdue support for hierarchical packages. InstallerMaker 5.3 solidifies and strengthens our earlier work through code optimizations and completion of full scripting support.

InstallerMaker's Network or Internet support includes both FTP and HTTP file transfers. Developers create "Network Install" tasks for the files to be retrieved. These tasks consist primarily of the name of the host to retrieve the file from and the path leading to the file. Developers may optionally choose to configure their installers to automatically decode BinHex and MacBinary files and decompress StuffIt .sit files and .zip files. It's very easy to use, flexible, and has powerful installer technology.

StoneTable

You thought it was **just** a replacement
for the List Manager ?

We lied, it is **much** more !

Tired of always adding just one more feature to your LDEF or
table code ? What do you need in your table ?

Pictures and Icons and Checkboxes ?
adjustable columns or rows ?
Titles for columns or rows ?
In-line editing of cell text ?
More than 32K of data ?
Color and styles ?
Sorting ?
More ??

How much longer does the list need to be to make it worth
\$200 of your time ?

See just how long the list is for StoneTable.

Make StoneTable part of your toolbox today !

Only \$200.00

MasterCard & Visa accepted.

StoneTable Publishing
More Info & demo Voice/FAX (503) 287-3424
<http://www.teleport.com/~stack> stack@teleport.com

The 5.0 release focused on the functionality most requested by our customers. InstallerMaker has always required conditions, package assignments, and actions to be set only at the root level of archives, which meant items in folders would always receive the same settings as their root parents. This made it difficult to create a single installed folder with items installed based on many different criteria. Developers had to use folder factoring, a process by which multiple folders of the same name in an archive are merged together during installation, to build these installers. Version 5.0 blew away that barrier, so now every archive item may be individually assigned different conditions, packages, and actions.

Hierarchical packages were also a big part of the 5.0 release. We worked hard to make it easy to configure for developers and easy to use for their customers. We wanted to preserve the original functionality, for developers not wishing to use the new support, while extending the feature set out for those who did. We also shied away from changing the design toward things we didn't like about other installers we have seen, which we considered less intuitive and thus less user friendly to use. Specifically, we wanted to keep the descriptions of packages displaying in the main installer window automatically when a user clicked on a package name. We also added a selection button with a question mark on it next to each package in a package list. We wanted to get users connected with information as efficiently as possible.

InstallerMaker 5.3 is a developer's dream release, especially if an automated build system is the goal. This release fills out all the nooks and crannies of automatic generation of installers through AppleScript. InstallerMaker 5.3, with the new native AppleScript support in Mac OS 8.5, is a powerful combination for supporting an automated build system. We also provide a FileMaker Pro database example, used in conjunction with AppleScripts, demonstrating a complete automated build system. Aladdin uses a derivative of that example for our own automated build system for our products. The database allows product managers to file installer requests that are easily turned into automatically generated installers for Aladdin products. It's a really fluid process and quite elegant in how it handles our installer needs.

Dave: I've heard about a product called FlashBack. What can you tell me about it?

Peter: Aladdin FlashBack provides unlimited Undo's for any application with instant access to all previous

versions of a document. With Aladdin FlashBack, you can create, compose, edit and save documents in any application without the fear of ever losing their work. It recovers previously saved versions of any FlashBack-protected document, even if the original file is lost, damaged, overwritten or erased. It tracks and records only the changes made to a document each time it is saved, eliminating the need for "Save As..."

To "FlashBack" a document, the user simply drags and drops the file into the Aladdin FlashBack window. Every time the file is saved, FlashBack uses a "differencing engine" to compare the current file to the previously saved version and captures and records the changes. Only the changes between these two files are recorded and saved by the FlashBack application (under the same title with a time and date stamp), so disk space is reduced compared to the conventional "Save As..."

The FlashBack application window lists all the tracked files and their versions. To retrieve a previous version, the user simply double-clicks on the selected version, or drags it to the desktop. FlashBack launches the file's application (if it is not already running) and combines the current document with the difference files to generate the version the user selected. The file is recreated exactly as it was at the time and date indicated.

FlashBack maximizes efficiency by letting the user set the frequency with which it records changes. The user can configure FlashBack to track changes whenever a document is saved, or whenever it is closed. With programs that have an auto-save function, FlashBack can be set to save changes every few minutes, instead of every instant, since many auto-save programs save so frequently. Settings can be made on a default basis, or tailored to a file or application.



**Want to get Mac OS developer
news delivered to you?**

Subscribe to MacDev-1™

**A subscription form available
through a link on the MacTech®
home page at
<<http://www.mactech.com>>**

by John C. Daub, Austin, Texas, USA

The Ultra-Groovy LString Class

An introduction to the PowerPlant-way of working with strings

HELLO, WORLD!

Most software developers have been working with strings since they wrote their first program — that's what "Hello World!" is all about. To facilitate working with strings many libraries of utility functions have been written. Some people have their home-brewed string libraries, while others utilize third party libraries or use the libraries provided in Standard C and C++. These solutions are well and good, but they do not always translate very effectively into the world of Mac OS. Many of these offerings, like the Standard C and C++ libraries, are written to work with C-style strings — a sequence of characters terminated by a null character. But due to the Mac OS Toolbox's Pascal heritage, strings on the Mac are Pascal-style strings — a sequence of characters preceded by a *length byte* which stores the length of the string. Consequently, a string library designed to work with Pascal-style strings is more valuable to the Mac OS software developer. Enter PowerPlant's LString class.

LString is a C++ abstract base class for working with Pascal-style strings. It provides means for converting strings to numbers and numbers to strings; obtain strings from numerous sources (characters,

other strings, resources); copy, append, and compare strings; find, insert, remove, and replace substrings; and does so in a manner that respects the boundaries of Pascal-style strings, takes advantage of the C++ language, makes your code easier to read and understand, and most of all is easy and logical to use. The inherited classes LStr255, TString, and LStringRef provide the concrete means for working with strings in a manner very familiar to Mac OS software developers, so the learning curve for LString is not a steep one. And although LString is a PowerPlant class it can be used independent of the rest of PowerPlant; feel free to use it in all of your Mac OS software development efforts.

If LString sounds good to you, then please read on. In this article I hope to provide you with a good overview of what LString provides, and illustrate how it will make your software development efforts easier. Along the way I'll also show you a few of the really cool and groovy power-user features of LString. But don't worry! Those power-user features are so easy to use that you'll be an LString-Warrior before you know it!

FIRST, SOME BACKGROUND

Before I go any further, I want to make sure that we're on the same grounds of understanding just what a string is, and what the differences are between the various types of strings. If you already understand what the different types of strings are, feel free to skip ahead to the next section.

Simply put, a *string* is a sequence of characters. This sentence is an example of a string.

A *C-style string* is an array of characters (`char*`). The internal representation of the string has a null character (`'\0'`) at the end, to signal the end of the string, so the amount of storage required for the string is one more than the number of characters in the string. Due to the use of the null terminator, there is theoretically no limit to the length of a C-style string, but you must scan the entire string to determine the length of the string (the C standard library function `strlen()` can be used to determine the length of the string). So the string "Hello, World!" contains thirteen characters, but at least fourteen bytes must be allocated to hold that string.

John C. Daub can't think of anything snazzy to put in his bio other than 311's a great band to listen to while writing articles. If you'd like to contact John, you can do so at hsoi@metrowerks.com.

A *Pascal-style string* is also an array of characters (`unsigned char*`). Like a C-style string, a Pascal-style string also requires one more than the number of characters in the string for storage. However unlike a C-style string (and this is the defining and differentiating characteristic) there is no null terminator at the end of the string; instead, the first byte of the string is used to store a count of the number of characters in the string. You do not need to scan the entire length of the string to determine its length — you can merely examine the first byte (*length byte*) of the string. Also unlike a C-style string, a Pascal-style string does have a limit; this limit is determined by the number of bytes allocated in the `unsigned char*` array (e.g. `unsigned char[256]` would allow a string up to 255 characters). As a Pascal-style string “Hello, World!” still contains thirteen characters and also needs at least fourteen bytes for storage, but the internal representation of the string is different. Finally, when specifying a Pascal-style string constant, they are typically prefaced by a `\p` to tell the compiler to treat this string as a Pascal-style string (e.g. `\pHello, World!` would be the actual way to represent our example string as a Pascal-style string). Figure 1 illustrates how the internal storage of the “Hello, World!” string differs as a C-style string and a Pascal-style string.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
H	e	l	l	o	,		W	o	r	l	d	!	\0
13	H	e	l	l	o	,		W	o	r	l	d	!

Figure 1. String internal representations.

In **Figure 1**, each column represents a byte in the array (note we start at zero, like any array). The top row is the byte count, the middle row is the C-style string, and the bottom row is the Pascal-style string. Notice that both string styles require the same amount of storage (fourteen bytes). The C-style string places a null terminator in the last byte (although it looks like two separate characters `\0` is just a special single character). When the string is read in, characters are read one by one until the null terminator is reached. If we wanted to find the length of the C-style string we would scan the string, starting from the beginning, counting each character as we go along until we reach the null terminator (this is what `strlen` does). The Pascal-style string places the length of the string in the first byte of the array (in this case the number thirteen, not “13” as a string or character). When the string is read, the first byte is checked for the length, and then exactly that many characters are read. If we wanted to find the length of the string, we check the first byte (`string[0]`).

Both styles of strings have their strengths and weaknesses. C-style strings can be of an arbitrary length whereas Pascal-style strings tend to have a more fixed size. C-style strings also have more overhead involved in determining the length of the string (a function call and walking the entire string), but you can find the length of a Pascal-style string with a simple check of the length byte.

On the Mac OS, most strings are Pascal-style strings (since the OS and toolbox were originally written in Pascal). To more easily represent and identify strings, and provide a simple means

for defining strings of different lengths, `MacTypes.h` (formerly `Types.h`) typedefs some arrays of `unsigned char`'s. The most popular version is a `Str255`, which is an array of 256 `unsigned char`'s. There are others (`Str63`, `Str32`, `Str15`, `StringPtr`, `ConstStr255Param` — see `MacTypes.h` for a complete listing), and although `Str255`'s are the most commonly used, the discussions in this article apply to any of these type(def)s. The Mac OS can also have raw text runs (e.g. ‘TEXT’ resources, `TERec`, etc.), and you can use C-style strings as well. The only time you are “forced” to use a Pascal-style string is when you interact with the Toolbox. If you wish to use C-style strings otherwise, you are very welcome to do so, especially if you are more familiar with the C Standard Library. Furthermore, the Toolbox provides some Pascal to/from C string conversion routines (`c2pstr` and `p2cstr` are in the Universal Header `TextUtils.h`), and even provides C-glue routines that are glue routines to Toolbox functions that take C-style strings as arguments instead (look for the symbol `CGLUESUPPORTED` throughout the Universal Headers).

But as most Mac OS software development sooner or later requires you to interact with the Toolbox, most people find it easier to use Pascal-style strings all the time (also avoids the constant overhead of back and forth conversions). Due to this preference is why PowerPlant offers a great solution in `LString`. Let's now take a look at what exactly `LString` has to offer.

WHAT'S IN THERE?

The `LString` class (and all of the classes and material I'll discuss here) can be found within the PowerPlant folder of the Metrowerks CodeWarrior Professional product. Specifically within the files `LString.cp` and `LString.h`. If you own CodeWarrior Professional, go ahead and open up those two files and look over them. I don't expect you to necessarily understand what's in there right now, but give them a look over so you can know what I'm talking about. You might also find it handy to refer to the sources as I refer to various parts of it so you can see how all of this fits together. For reference, I am writing this article using CodeWarrior Pro 4 and a version of `LString` that should be part of PowerPlant v1.9.3 (which will be released as public netborne update to Pro 4, and should be available by the time you read this article).

LString

`LString` is an abstract base class for working with Pascal-style strings. It defines almost all of the functionality that one would need for working with Pascal-style strings. `LString` also works to take advantage of the C++ concept of overloading. First many of the methods are overloaded to work with many data types (all built-in data types, both signed and unsigned, including floating point types; C-style strings; Pascal-style strings; `FourCharCode`; a `Handle` to text; other `LString` objects). Second, many of the methods are also offered as operator overloads, were logical, to make it easier for you to utilize these methods in your code. For example, `operator +=` is the same as the `Append()` method. `LString` also defines some public methods as static so that non-`LString` objects can take advantage of some commonly needed string manipulations like copying and appending.

LString is designed to be clean, efficient, and work within the boundaries of the Mac OS and the Pascal-style strings utilized therein. As noted earlier, `MacTypes.h` defines different types of Pascal-style strings, like a `Str255` or a `Str63`. Due to this ability to vary in storage length, any time length of string and/or storage is relevant to the functionality, LString always requires this information be provided, but typically also provides a default towards a `Str255` as this is the most commonly used type. Furthermore, since Mac OS Pascal-style strings have 255 characters as the upper limit for strings (hence the `Str255`), LString always enforces this upper boundary where appropriate. If utilized correctly, LString should alleviate fears and eliminate the problem of array boundary overflows.

Although PowerPlant is a Mac OS-only framework, it does try to do what it can to aid those that might be using PowerPlant in a non-Mac context such as porting your Mac OS-based application to Windows. Some LString methods are only available under Mac OS (due to their nature), and techniques are provided for development environments that might not fully understand Mac OS conventions. For example, the traditional way to obtain the length of the string is to directly check the length byte. But since not all development environments support this method of access (of course CodeWarrior does), the `LString::Length()` method is provided as a certain means of always obtaining the length of the string.

As an implementation note, there is only one data member in LString: `mStringPtr`. `mStringPtr` is a `StringPtr` which points to your Pascal-style string. Note that the actual storage for the string is **not** part of LString. Without this sort of design, neither `TString` nor `LStringRef` could exist (I'll discuss these classes later in the article). So does this mean that you always have to allocate your own storage for your string? Not at all, as this is the purpose of `LStr255`.

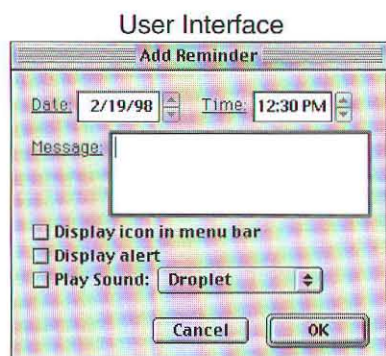
LStr255

`LStr255` inherits from `LString`. You can find the declaration of `LStr255` in `LString.h` and definition in `LString.cp`. If the name of the class looks familiar to you, this is intended. The design of `LStr255` is to replicate the functionality of a `Str255`, but with lots of extras. The great part is an `LStr255` can be used almost anywhere a regular `Str255` is used and in the same manner. Furthermore an `LStr255` inherits all of the functionality of `LString`, plus it provides the actual storage for the string itself (the `mString` data member) so you do not have to.

Listing 1: Use of Str255

Obtain and concatenate two strings
To explain a convention used in all listings: any function call preceded by the *unary scope resolution operator* (`::`) specifies this symbol is located in the *global namespace* — the same place where the Toolbox symbols are located. These are Toolbox calls. Feel free to look them up in *Inside Macintosh* on the Apple WWW site for more information and detail.

AppMaker – More than a GUI Builder



AppMaker generates Get and Set functions to access your data items and generates code to call the Get and Set functions when UI items are changed, and to update UI items when data items are changed. Also generates file I/O code.

**B•O•W•E•R•S
Development**

P.O. Box 929, Grantham, NH 03753 • (603) 863-0945 • FAX 863-3857
bowersdev@aol.com • <http://members.aol.com/bowersdev>

Also helps you write the application logic, helps you separate UI code from functional code, helps you connect UI items to functions.

AppMaker makes it faster and easier to make a Mac application. Just point and click to declare your data structures, and to design your user interface. AppMaker generates resources and source code to implement your design.

Application Logic

```
ReadFile()
GetReminders()
    ->AddItem(newItem)
SetMessage(...)
GetYearMonthDay()
SetHourMinute(...)
```

Users describe the AppMaker-generated code as "human professional quality." AppMaker generates C, C++, or Pascal for Apple's Appearance Manager and C++ for CodeWarrior's PowerPlant.

AppMaker is just \$199 from Developer Depot (www.devdepot.com) or from Bowers Development.


```
// Declare a string and initialize it
Str255 theString = "\pHello, World!";

// Append another string, obtained from a STR# resource
Str255 appendString;
::GetIndString(appendString, 128, 1);

// Ensure we don't overrun our boundaries
SInt16 charsToCopy = appendString[0];
Size theStringSize = sizeof(theString);

if ((theString[0] + charsToCopy) > (theStringSize - 1)) {
    charsToCopy = theStringSize - 1 - theString[0];
}

// Append the string
::BlockMoveData((Ptr)&appendString[1], theString +
    theString[0] + 1, charsToCopy);

// Reset the length byte
theString[0] += charsToCopy;

// Draw the string
::DrawString(theString);
```

Listing 1 demonstrates a typical set of actions that one might perform on a string. All of that work and code just to concatenate two strings. Listing 2 shows the same code but written using LString. Don't worry if you do not understand everything that is going on in Listing 2 because I'll cover it all in coming sections.

Listing 2: Use of LStr255

Obtain and concatenate two strings

```
// Declare a string and initialize it
LStr255 theString( "\pHello World" );

// Append another string, obtained from a STR# resource
LStr255 appendString(128, 1);

theString += appendString;

// Draw the string
::DrawString(theString);
```

As you can see comparing Listing 2 to Listing 1, the same result is accomplished but Listing 2 contains a lot less code and is much easier to read. This just scratches the surface of what LString can do.

Two usage notes. First, when you are using LStr255 objects and viewing them in a debugger, you will actually see two instances of your string data within the LStr255 object. One of these will be from mStringPtr and the other from mString. Why do you see two strings and why are they always in sync? Why not just have one string? Remember that mStringPtr is just a pointer to your actual storage for your string, mString; rather, mStringPtr points to mString. This is how things are designed and it's all OK, don't worry about it. However if they are not in sync, you might want to investigate as this could be signaling a problem. Second, beginning with the next section and continuing for the remainder of the article, I will be using LStr255 objects in all code listings because LStr255 is the most commonly used type of LString. Do remember that what applies in those listings will generally apply to other LString derivatives as well, like TString and LStringRef.

TString

TString is a templated version of LString. It allows you to utilize the power and functionality of LString upon any Pascal-style string type (Str255, Str63, Str15, etc.). Use of the class is exactly like using LStr255, except that you instantiate your object through normal C++ template instantiation mechanisms. Listing 3 is a rewrite of Listing 1 using TString. Note as well its similarity to Listing 2.

Listing 3: Use of TString

Obtain and concatenate two strings

```
// Declare a string and initialize it
TString<Str63> theString("\pHello World");

// Append another string, obtained from a STR# resource
TString<Str255> appendString(128,1);

theString += appendString;

// Draw the string
::DrawString(theString);
```

LStringRef

LStringRef is a newcomer to the LString family (it was introduced in PowerPlant v1.9.1, post CodeWarrior Pro 3). LStringRef provides you with a means to obtain the functionality of LString and use it to manipulate a string that you do not have ownership of. Unlike LStr255 and TString, LStringRef does **not** contain storage for the string itself; mStringPtr points to a string allocated elsewhere. For example, this can be used to change a string that is part of some data structure. Listing 4 demonstrates the use of LStringRef to change the filename in an FSSpec from whatever it originally was to "Hello File! copy".

Listing 4: Use of LStringRef

```
// Obtain our file spec
FSSpec theFileSpec;
theLFile->GetSpecifier(theFileSpec);

// Create the LStringRef and have it point to the filename
LStringRef fileName(sizeof(theFileSpec.name),
    theFileSpec.name);

// Change the filename entirely
fileName = "Hello File!";

// Actually it's a copy
fileName += " copy";
```

BASIC FEATURES

Now that you've been introduced to the LString family and seen a little of what LString can do, it's now time to dig through the lines of code in LString.cp and LString.h to really see all that LString has to offer. By the way, do make sure you also read through LString.h as many of LString's functions are declared inline in the header file.

Object Construction

If you look through the LString sources, you'll notice that there is only one LString constructor. That's all that LString needs as it performs the core setup of the class's functionality. Where

the constructors are really needed is within the subclasses. Look at the number of constructors there are for `LStr255`! And look what you can do with all of those constructors as well. You can create your `LStr255` from: another `LStr255`, another `LString`, a Pascal-style string, a C-style string, an arbitrary string of text, a single character, data within a `Handle`, from a `'STR'` or `'STR#'` resource, from a long or short integer, a `FourCharCode`, and from floating point values (long double). Wow! That covers just about every and any way you might want to create your `LStr255` object.

If you look at the implementation of most of those `LStr255` constructors, you'll see many of them call the `Assign()` method. `Assign()` assigns the given value to the `LString` object. Just as the constructors use `Assign()`, you are free to use `Assign()` as well when you wish to assign a value to your `LString` object. Furthermore, as a logical use of C++, you can see in `LString.h` that the assignment operator (`operator =`) has been overloaded with just as many combinations for those of you that prefer the use of operators in logical situations. Listing 5 demonstrates the many ways `Assign()` can be used.

Listing 5: The Many Faces of Assign()

Each line group produces the same result of creating an `LStr255` object and giving it a string of "Hello, Assign!".

```
// Create and assign by initialization
LStr255  initString("Hello, Assign!");
// Create and assign by Assign()
LStr255  assignString;
assignString.Assign("\pHello, Assign!");
// Create and assign by assignment
LStr255  assignmentString = assignString;
```

String Manipulation

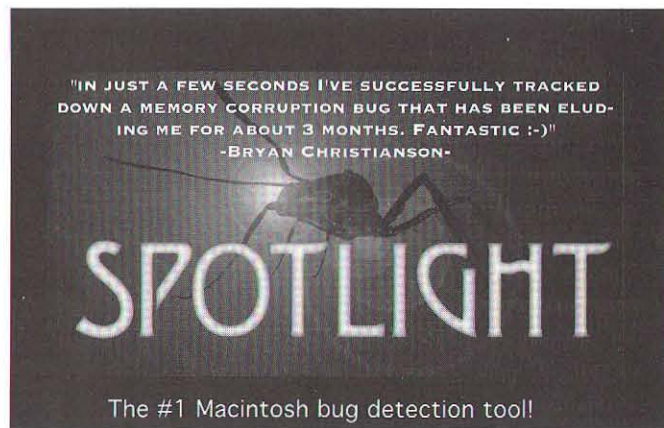
Now that you have created your `LString` and given it some text, you will probably want to do something with the text in that string. This functionality is what `LString` tends to be used for most, and within this subset lies the use of `Append()`.

`Append()` appends a given value to an `LString` object. What can be appended is again just about anything (see `LString.h` for a complete list). As with `Assign()` and `operator =`, so with `Append()` and `operator +=`. You'll probably find yourself using `operator +=` a great deal as it's so simple to type and makes your code so much easier to read. One issue to note with `operator +=` (and perhaps other overloaded functions in `LString`) is that depending what you are trying to append you may receive compiler errors about an ambiguous access to an overloaded function. For instance, the following line of code will generate this error because the compiler cannot determine if you intend 1 to be treated as `char`, `unsigned char`, `long`, or `short`:

```
theString += 1;
```

If you receive this compiler error, all you need to do to resolve it is apply a `static_cast` (or C-style cast) which will tell the compiler explicitly how you wish the value to be treated:

```
theString += static_cast<Int32>(1);
```



"IN JUST A FEW SECONDS I'VE SUCCESSFULLY TRACKED DOWN A MEMORY CORRUPTION BUG THAT HAS BEEN ELUDING ME FOR ABOUT 3 MONTHS. FANTASTIC :-)"
-BRYAN CHRISTIANSON-

SPOTLIGHT

The #1 Macintosh bug detection tool!

- Detect memory leaks automatically
- Instruction level bounds checking
- Validate 400 Mac Toolbox calls
- Pinpoint stale handle usage

- Integrated with all Mac debuggers
- Debug shlibs and stand alone code
- Faster than ever before

**FREE
DEMO**

Finalist
Macworld
Editors' Choice



941.795.7801 Fax: 941.795.5901
www.onyx-tech.com sales@onyx-tech.com

In addition to member functions for appending, `LString` also provides some global string addition operators (`operator +`). They are **not** `LString` member operators, but do act upon `LString` objects. These operators concatenate `LString` objects with: another `LString` object, a pointer to a string, or a character. The result of the addition is an `LStr255` object. You can find these operators declared about mid-way through `LString.h` and defined near the bottom of `LString.cp`.

Finally, what would string manipulation be if you could not compare strings! You will find in `LString.h` a boat-load of global comparison operators that let you do just about any sort of string comparison you can think of: equals (`operator ==`), not equals (`operator !=`), greater than (`operator >`), less than (`operator <`), greater than or equals (`operator >=`), and less than or equals (`operator <=`). Listing 6 demonstrates the many ways you can manipulate strings with `LString`.

Listing 6: String Manipulation

```
// Create our string, "Hi"
LStr255  theString("Hi");

// Whoops! We forgot the punctuation
theString += '!';

// Create a new string from 2 smaller strings
LStr255  newString;
newString = theString + "\p My name is John.";

// Is our newString the same as our old string?
```



```
// If so, beep.
if ( newString == theString ) {
    ::SysBeep(2);
}
```

Utilities

Rounding out the basic features of `LString` are some utility functions. They fall into two groups: object-related and public.

The object-related utilities are utilities that are class member functions and only behave in relation to their associated `LString` object. These are methods like: `Length()`, which returns the string length as an `UInt8`; `ShortLength()` which returns length as an `Short`; `LongLength()` which returns length as an `Long` (the different return types are to ease places where you need the string length but do not wish to typecast to match a function parameter or avoid an implicit arithmetic conversion); `GetMaxLength()`, to return the maximum length the string can be; `TextPtr()` and `ConstTextPtr()` to return a `Ptr` (or `const Ptr`) to the raw text (avoids those ugly `(Ptr)&theString[1]` situations, as was used in Listing 1); `operator []` to allow access to individual characters in the string, just like you could with a basic array. Listing 7a illustrates the use of these utilities.

Listing 7a: Utilities

```
LStr255 theLStr255("\pYea");
TString<Str255> theTStr255; // LString::LString() properly
                           // initializes to null.

// Manually copy the text out of the LStr255 into a Str255
::BlockMoveData(theLStr255.ConstTextPtr(),
                (Ptr)&theTStr255[1],
                theLStr255.LongLength() );

// Reset length byte
theTStr255[0] = theLStr255.Length();
```

The public utilities of `LString` are a small group of static methods, publicly available for calling by pretty much anyone anywhere in your program (within reason, of course). These methods handle the most common string manipulations like: `CopyPStr()`, to copy one Pascal-style string into another Pascal-style string (e.g. `Str255` to `Str255`); `AppendPStr()`, for a static version of `LString::Append()`; `CStringLength()` for obtaining the length of a C-style string with an upper limit of 255; `FourCharCodeToPStr()` and `PStrToFourCharCode()` for converting a `FourCharCode` (y'know, 'TEXT', 'PICT', 'PPob') to a Pascal-style string and back again; and `FetchFloatFormat()` and `StringToLongDouble()` to aid in some quick floating-point conversions. Listing 7b presents a display of these utilities' features.

Listing 7b: Static Utilities

```
// Create a Pascal-style string and assign it a value
Str255 herString;

LString::CopyPStr("\pMary", herString);

// Turn it into our app's signature
FourCharCode herSig;
LString::PStrToFourCharCode(herString, herSig);
```

```
// Clear original storage
herString[0] = 0;

// And back again
LString::FourCharCodeToPStr(herSig, herString);

// Add the rest
LString::AppendPStr(herString, "\p Victoria");
```

POWER FEATURES

In addition to the basic functionality's discussed above, `LString` has some features for the power-user that help to round out the practicality of the class. If you consider yourself a beginner to `LString`, you should still give these power-features a look over as using these power-features is not difficult, and you can be a power-user in no time. The substring manipulations you may or may not use every day, but the "typecasting" abilities you'll find indispensable.

`LString`'s ability to manipulate substrings provides you with the capability to perform substring searches, copies, and changes to that substring. In fact, you might even be able to create your own *Find* dialog using these features. The search functionality in `LString` allows you to: `Find()`, locate where a substring starts within the string; `ReverseFind()`, locate the position of a substring starting from the end of the string; to see if a string `BeginsWith()` or `EndsWith()` a specified string; find where the string starts within another string either from the beginning, `FindWithin()` or from the end, `ReverseFindWithin()`. In performing these searches, a comparison of text must of course be done. `LString::ToolboxCompareText()` is provided as the default mechanism for comparing the text; it uses the `ToolboxCompareText()` routine. If you do not wish to use the default comparison mechanism, you can specify your own `CompareFunc` via `SetCompareFunc()` to change it to whatever you would like. Of course after you have found your substring, you might want to do something with it. If you would like you can `Insert()` a string into your `LString` object, `Remove()` a substring from your object, or `Replace()` one substring with another. Or if you would just like to make a copy, `operator ()` has been overloaded to provide you with an easy way to perform this copy. Listing 8 illustrates `LString`'s substring manipulation features.

Listing 8: Substring Manipulation

```
// Create the base string
LStr255 theString("Get up. Stand up for your rights.");

// Ensure the compare function is what we need
theString.SetCompareFunc(LString::ToolboxCompareText);

// Find a target string
StringPtr targetString = "\pStand up";
UInt8 targetLength = targetString[0];

UInt8 index = theString.Find( (Ptr)&targetString[1],
                             targetLength );

// Make a copy of the target string
LStr255 targetCopy(theString(index, targetLength));
targetCopy += "\p. ";

// Insert the copy into the base string
theString += char_Space;
theString.Insert( targetCopy, index );
```


What I think is one of the niftiest features of `LString` is, as I like to call them, the *typecasting operators*. These are a series of operator overloads that fake the appearance of a typecast to allow your `LString` object to be flexibly used in many situations. You can find the operators listed near the top of the `LString` declaration in `LString.h`. These operators include: `StringPtr`, `ConstStringPtr`, `SInt32`, `double`, `long double`, and `FourCharCode`. When one of these operators is applied to the `LString` object, it converts the string into the “requested type” and returns the conversion. That is, apply operator `SInt32` and the string is converted to a long integer and that long integer returned. The way to apply these operators is just like a typecast, but it's not really a typecast; it's technically an application of that operator, but the implementation and application of that operator simulates a typecast for ease of use and improved code readability. Furthermore, when attempting to resolve types, the compiler can implicitly apply these operators to the `LString` object for you. Look back at Figure 2 and notice that we passed our `LStr255` object directly to `DrawString`? Listing 9 should clarify how all of this works.

Listing 9: Typecasting Operators

```
// Create our string (FIXEDDECIMAL comes from fp.h)
LStr255 theString(3.14, FIXEDDECIMAL, 2);

// Convert it and do some math (the static_cast technically
// isn't necessary, but here for illustration. Note that it
// looks like a typecast, but if you watch the code execute
// you'll step into operator double()).
double number = static_cast<double>(theString);
number *= 2;

// Convert back to a string. Note there is no operator =
// for floating point variables. This is because of the
// need for extra information.
theString.Assign(number, FIXEDDECIMAL, 2);

// Draw onscreen. Note the implicit operator call to
// operator StringPtr.
::DrawString(theString);
```

PowerPlant uses this technique of *typecasting operators* throughout the framework as a handy way to allow your objects to be treated as more basic Toolbox types for increased flexibility and seamlessness between PowerPlant, the Toolbox, and your code. See the use of operator `Handle` and operator `Ptr` in `UMemoryMgr` classes for another set of examples. Also, if you have access to the Scott Meyers book *More Effective C++*, read *Item 5: Be wary of user-defined conversion functions*. Scott labels the above technique *implicit type conversion operators* and points out the various strengths and weaknesses of the above technique.

IS THERE A DOWNSIDE?

As great as `LString` is, there are certainly a few downsides. There may be others, but these are the big ones in my book. First, if you don't like Pascal-style strings or perhaps they're not as useful as a C-style string would be in a given situation, then of course `LString` would fall under the same roof. But if you use and/or like Pascal-style strings then this isn't much of an issue. Second, this code is C++. If you need Pascal-style string helper

functions in C or another language then `LString` will not be of much use to you. Third, `LString`'s are C++ objects. As an object it will have some additional overhead and memory requirements, at least compared to plain arrays (e.g. `LStr255` vs. `Str255`). Furthermore because `LString`'s are objects you cannot place them inside `TArray`'s, except, as with all objects and `TArray`, as pointers to the object allocated via `new`. Due to this extra work, it's probably easier to just store plain `Str255`'s in a `TArray` instead. But do not fret! If you must use `Str255`'s you can still gain the features and power of `LString` via the `LStringRef` class applied to your `Str255`'s.

GIVE IT A TRY

If you haven't already, give `LString` a try. The code listings should all function as compilable snippets. And since `LString` is mostly independent of the rest of PowerPlant (you may need to add the `PP_Constants.cp` file to your project as well), you can easily drop `LString.cp` into a basic Toolbox or console stationery and give them a try. Step through each listing in the debugger (make sure inlining is set to “Don't inline”) and watch how things work. Watch where you go, how that pertains to the section material being discussed, and how they all fit into the larger picture. Just play around and experiment.

I hope that I've been able to give you a good introduction to the `LString` class and it's family of subclasses and utilities. It provides the Mac OS software developer with a fairly simple yet powerful class for working with Pascal-style strings. With the ability to create strings from almost any source, manipulate the contents of that string, and convert it for use in just about any situation, `LString` is a tool worth having in your programmer's toolbox.

I hope you find `LString` to be as ultra-groovy as I find it to be. But just in case, I'll borrow one from Dennis Miller: “Of course that's just me. I could be wrong.”

BIBLIOGRAPHY

- * Kernighan, Brian W. & Dennis M. Ritchie. *The C Programming Language*. 2nd edn. Prentice Hall, Englewood Cliffs, New Jersey. 1988.
- * Metrowerks Corporation. *PowerPlant Essentials*. 1998.
- * Meyers, Scott. *More Effective C++: 35 New Ways to Improve Your Programs and Designs*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1996.
- * Stroustrup, Bjarne. *The C++ Programming Language*. 3rd edn. Addison-Wesley Publishing Company, Reading, Massachusetts. 1997.
- * <<http://www.apple.com/developer/>>
- * <<http://www.metrowerks.com/>>
- * <news:comp.sys.mac.oop.powerplant>
- * <news:comp.sys.mac.programmer.codewarrior>



by Steve Sisak <sgs@codewell.com>

C++ Exceptions in Mac OS Code

Modern C++ offers many powerful features that make code more reusable and reliable. Unfortunately, due to its UNIX roots, these often conflict with equally important features of commercial quality Mac OS code, like toolbox callbacks, multi-threading and asynchronous I/O. C++ Exception Handling is definitely an example of this. In this article, we will describe some techniques for using C++ Exceptions in commercial quality MacOS C++ code, including issues related to toolbox callbacks, library boundaries, AppleEvents, and multi-threading.

WHAT ARE EXCEPTIONS?

Exception Handling is a formal mechanism for reporting and handling errors which separates the error handling code from the normal execution path. If you are unfamiliar with how C++ exceptions work, you may want to check out Chapter 14 of "The C++ Programming Language" by Bjarne Stroustrup or any of the other excellent texts on the topic.

Why are exceptions necessary? "Exceptions cannot be ignored"

— Scott Meyers

One of the problems in designing reusable code is deciding how to communicate an error that occurs deep within a library function back to someone who can handle it. There are several conventional ways for library code to report an error, including:

- Terminating the program
- Returning an error result
- Setting a global error flag
- Calling an error function
- Performing a non-local goto (i.e. longjmp)

While terminating a program as the result of an error in input may be considered acceptable in the UNIX world, it is generally not a good idea in software you plan to ship to human users.

Returning an error or setting a flag are somewhat better, but suffer from the fact that error returns can be (and often are) ignored, either because the programmer was lazy or because a function that returns an error is called by another which has no way to report it. Both of these methods are also limited in the amount of information they can return. Return values must be meticulously passed back up the calling stack and global flags are inherently unsafe in a threaded environment because they can be modified by an error in a second thread before the first thread has had a chance to look at the error.

Calling an error handler function is reliable, but while the function may be able to log the error, it must still resort to one of the other mechanisms to handle or recover from the error.

This leaves non-local goto, which is basically how exceptions are implemented — except with formal support from the compiler. C++ exceptions extend setjmp/longjmp by guaranteeing that local variables in registers are handled properly and destructors for any local objects on the stack are called as the stack unwinds.

Because an exception is an object, it is possible for a library developer to return far more information than just an error code.

What's wrong with C++ exceptions?

In a nutshell: Lack of Standardization.

Like many aspects of C and C++, the implementation of exceptions has been left as an implementation detail to be defined by compiler vendors as they see fit. As a result, it is never

Steve Sisak lives in Cambridge, MA, with two neurotic cats, and ten Macintoshes. Steve referees Lacrosse, plays hockey, and enjoys good beer and spicy food. Products he has worked on include The American Heritage Electronic Dictionary, PowerSecretary, Mailsmith, MacTech's Sprocket, and several others. He currently makes his living making applications scriptable and developing MacOS USB drivers.

safe to throw a C++ exception from a library that might be used by code compiled with a different compiler (or a different version of the same compiler, or even the same version of a compiler with different compile options).

As a result of this:

- Exceptions must not be thrown out of a library.
- Exceptions must not be thrown out of a toolbox callback.
- Exceptions must not be thrown out of a thread.

Each of these cases can fail in subtly different ways:

In the first case, there is no guarantee that both compilers use compatible representations for exceptions. The C++ standard does not define a format for exceptions that is supported across multiple compilers—C++ exceptions are objects and there is no standard representation for C++ objects that is enforced across compilers. This is also why it's not feasible to export C++ classes from a shared library.

IBM's System Object Model (SOM), used in OpenDoc and Apple's Contextual Menu Manager, solves this problem for objects quite robustly (even to the extent that it is possible to mix objects and classes implemented in different languages like C++ and SmallTalk), however, there are still additional issues which would require a "System Exception Model" as well.

As a platform vendor, Apple could have saved us a lot of work here by specifying a "System Exception Model" that all compiler vendors would agree to implement. In fact they began to implement an Exceptions Manager as part of the PowerPC ABI but it was left unfinished the last time the developer tools group was killed and Metrowerks took over as the dominant development environment—so we're stuck with the current state of incompatibility. Hopefully, now that Apple is working on developer tools again, we might finally see a standard.

Also, many Mac OS routines allow the programmer to specify callback routines which will be called by the toolbox during lengthy operations or to give the programmer more control than could be encoded in routine parameters. Unfortunately, because of the above limitations, it is not possible to throw an error from a callback and catch it in the code that called the original Toolbox routine.

This is because there is no way for the toolbox to clean up resources that may have been allocated before calling your function. In this case it is necessary to save off the exception data (if possible), return an error to the toolbox, and then rethrow the exception when the toolbox returns to our code. Of course, C++ provides no safe way to save off the exception currently being thrown for this purpose and RTTI does not provide enough access to extract all data from an object of unknown type, so again, we must roll our own.

Build great applications. Better. Cheaper. Faster!

CodeWarrior
Symantec
MacOS 8
System 7
68K
PowerPC
C/C++
Pascal

Good, fast and cheap.

They say pick two. We say you can have it all.

Tools Plus routines let you create user interface elements that work as soon as you make them. They're ready-to-use Macintosh application parts, just waiting to be assembled.

We include everything from tool bars and floating palettes to the coolest picture buttons and tons of 3D controls. **Tools Plus** works brilliantly with multiple complex dialogs, using resources and/or manual coding.

Use **Tools Plus** libraries alone or with our framework. It's compact yet immensely capable. Easy to learn, yet ready to grow with you.

**Good, Fast and Cheap.
Don't get two out of three.
Get Tools Plus.**

Tools Plus for

CodeWarrior Pro and Gold (68K and PPC)	\$249
CodeWarrior Bronze (68K)	\$199
Symantec Compilers (C/C++ & Pascal, 68K and PPC) ..	\$249
Symantec (THINK) C/C++ (68K)	\$149
THINK Pascal (68K)	\$149
THINK C/C++ & THINK Pascal (68K)	\$199

Compact
 Quick
 Easy

Tools Plus

LIBRARIES + FRAMEWORK

NEW TOOLS PLUS 4.0

MacOS 8

APPEARANCE SAVVY

"All in all, it's an incredibly rich collection of tools...If you are interested in developing applications that have 'quality' written all over them, then Tools Plus is for you."

MacTech MAGAZINE

Tools Plus

Macworld 1997 Editors' Choice Awards

Best Rapid-Development Tool

"...the routines are more compact and faster than anything you might write...Every element of Tools Plus is useful...a bargain compared with coding these routines yourself."

MW ★★★★★

Macworld

Orders and Enquiries:

Phone: (416)219-5628

Fax: (905)847-1638

WaterEdg@interlog.com

Free Evaluation Kit:

Available at our web site

<http://www.interlog.com/~wateredg>

Water's Edge Software

2441 Lakeshore Rd W.

Box 70022

Oakville, Ontario

Canada, L6L 6M9

There are a few toolbox managers that provide for error callback function that are not required to return. While you should be able to throw an exception from these callbacks, there are issues that you should be aware of. Specifically, some compilers implement so-called “zero-overhead” exceptions which use elaborate schemes of tables and tracing up the stack to restore program state without needing to explicitly save state at the beginning of a try block. Often this code gets confused by having stack frames in the calling sequence that the compiler did not generate, causing it to call terminate() on your behalf. (CodeWarrior's exceptions code also does this if you try to step over a throw from Jasik's Debugger—you can work around this by installing an empty terminate() handler.)

C and C++ have no notion of threading or accommodation for it. For instance, the C++ standard allows you to install a handler to be called if an exception is thrown and would not be caught, however you can only install one such handler per application. Further, it is technically illegal for this routine to return to its caller. So, there is no easy way to insure that an uncaught C++ exception will terminate only the thread it was thrown from rather than the entire program. (It is possible with globals and custom thread switching routines, but tricky to implement — I hope to have an example in the sample code by the time this is published.)

Interactions between threads and the runtime can also rear up and bite developers in even more interesting and subtle ways: For instance, in earlier versions of CodeWarrior's runtime, the exception handler stack was kept in a linked list, the head of which was in a global variable. As a result, if exceptions were mixed with threads and the programmer did not add code to explicitly manage this compiler-generated global, the exception stacks of multiple threads would become intermingled, resulting in Really Bad Things™ happening if anyone actually threw an exception.

What we need is a standard way to package an exception so it can be passed across any of these boundaries and handled or re-thrown without losing information.

How did AppleEvents get in here?

As any Real Programmer™ knows, good Macintosh programs should be scriptable (so users can do stuff the programmer didn't think of), and recordable (so that users don't have to have intimate knowledge of AppleScript to record some actions, clean up the result and save it off for future use).

You may also know that if you want to write a scriptable and recordable application and you're starting from scratch, the easiest way to do it is to write a “factored” application — where the application is split into user interface and a server which communicate with AppleEvents.

In a past life I've written about how using AppleEvents is a convenient way to make your application multi-threaded by using an AppleEvent to pass data from the user interface to a server thread. [MacTech Dec '94]

What you may not know (thanks to the fact that it's relatively hidden in the AppleScript release notes, rather than in Inside Macintosh or a Tech Note) is that AppleScript provides a relatively robust error reporting mechanism in the form of a set of optional

parameters in the reply of an AppleEvent which can specify, among other things, the error code, an explanatory string, the (AEOM) object that caused the error, and a bunch of other stuff.

Further, you may know that the AppleEvent manager provides a data structure that can hold an arbitrary collection of data (AERecord).

Putting this all together, if we define a C++ exception class which can export itself to an AERecord, we can both return extremely explicit error information a user of AppleScript (or any OSA language) and provide a standard format for exporting exception data across a library boundary. Also, since an AERecord can contain an arbitrary amount of data in any format, the programmer is free to include any information he wants in the exception — anything the recipient doesn't understand will be ignored.

Implementation Details

Following are some excerpts from an exception class and support code which do just this. Full source for a simple program using this code is provided on the conference CD. The exception mechanism is actually implemented as a pair of classes: Exception and LocationInCode and a series of macros which provide a reasonably efficient mechanism for reporting exactly where an error occurred and returning this information in the reply to an AppleEvent.

Using this mechanism, it is not only possible to throw an error across library boundaries, but also between processes or even machines.

Detection and Throwing Errors

The implementation of the Exception classes is divided between two source files: Exception.cp and LocationInCode.cp. The class Exception is the abstract representation of an exception. It has 2 subclasses: StdException and SilentException.

If you look at these two files, you'll notice that most of the functions that are involved in failure handling are implemented as macros in Exception.h which evaluate to methods of another class, LocationInCode — for instance, FailOSErr() is implemented as:

```
#define FailOSErr    GetLocationInCode().FailOSErr

#define GetLocationInCode() LocationInCode(__LINE__, __FILE__)

class LocationInCode
{
    LocationInCode(long line, const char* file) ...
    void Throw(OSStatus err);
    inline void FailOSErr(OSErr err) const
    {
        if (err != noErr)
        {
            // CW Seems not to be sign extending w/o cast
            Throw((OSStatus) err);
        }
    }
}
```

So that the expression:

```
FailOSErr(MyFunc());
```


Evaluates to:

```
LocationInCode(__LINE__, __FILE__).FailOSErr(MyFunc());
```

While this seems needlessly complex, there is a good reason for it, involving tradeoffs between speed, code size, and some “features” of the C++ specification.

Specifically, the obvious way to implement FailOSErr() is:

```
#define FailOSErr(err) if (err) Throw(err)
```

The problem here is that the macro FailOSErr() evaluates its argument twice. This means that, in the case of an error, MyFunc() will be called twice — clearly not what we want.

Here is one place that C++ can help us out — we can implement FailOSErr() as an inline function:

```
inline void FailOSErr(err)
{
    if (err != noErr)
    {
        Throw(err, __LINE__, __FILE__);
    }
}
```

Since C++ inline functions are guaranteed to evaluate their arguments exactly once, this solves our problem. Further, it makes it possible to have overloaded versions of FailOSErr which take different arguments, for instance a string to pass to the user, so you can write:

```
FailOSErr(MyFunc(), “Some Error message”)
```

The problem is that, once you implement this and try to access the file and line information, you will discover that, thanks to the way `__FILE__` and `__LINE__` are defined to work, all errors are reported as occurring in `Exception.h` — which is clearly less than useful. You would think that, in their infinite wisdom, the C++ standards committee would have updated the way that these macros work or provided a more robust mechanism for reporting the location of an error in code, but they didn't.

The solution presented here is a compromise—by instantiating the `LocationInCode` class from a macro, we insure that `__FILE__` and `__LINE__` evaluate to a useful location in the user's code, rather than in the exceptions library. Also, by using a class, we can reduce code size by allowing the methods of `TLocationInCode` to call each other without losing the actual location of the error.

An added benefit of this approach is that, in the future, we could replace the implementation of `LocationInCode` with one using MacsBug symbols or traceback tables in the code instead of relying upon the compiler macros.

Also, note that `FailOSErr()` and the constructor for `LocationInCode` are declared inline to maximize speed, but then call an out-of-line function (`Throw`) to minimize code size in the failure case.

Adding Information

At any point in handling an error you can add information to an `Exception` by calling `Exception::PutErrorParamPtr` or

`Exception::PutErrorParamDesc`. For instance, if you were in an `AppleEvent` handler and wanted to set the offending object displayed to the user, you could write:

```
try
{
    // whatever
}
catch (Exception& exc)
{
    exc.PutErrorParamDesc(kAEOffendingObject, whatever, false);
    throw;
}
```

These routines also take a parameter to tell whether to overwrite data already in the record — this is useful to ensure that the first error that occurred is the one reported to the user.

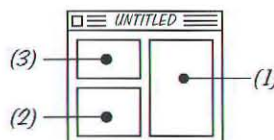
Insuring Errors are Caught

Because it's not safe to throw C++ exceptions across a library boundary, we need a mechanism to insure that all errors are trapped and properly reported. Unfortunately, unlike Object Pascal, we can't just call `CatchFailures()` to set up a handler — the code which might fail must be called from within a try block.

Also, because C++ effectively requires catch blocks to switch off of the class of the object thrown, and doesn't support the concept of ‘finally’ like Java, this master exception handler can end up containing quite a lot of duplicated code.

Quadrivio General Edit

Debug & Analyze Complex File Formats.



- 1) Define your format with familiar, C-like statements.
- 2) View, analyze, and edit data file in up to 5 columns.
- 3) View and edit file parameters. Also edit handle blocks and other data structures in memory.

Free!

Lite & demo versions
available on web site.

- ✓ **Quick insight** into data structure contents.
- ✓ **Easier** than studying MPW's dumpfile output.
- ✓ **Faster debugging** with clear display of data.
- ✓ **Saves coding time** with direct data editing.

Visit our web site for immediate download.

On-line (\$195) and boxed (\$249) versions available.
Also available from Developer Depot & SciTech International.

<http://www.quadrivio.com>

Quadrivio Corporation info@quadrivio.com
1563 Solano Avenue #360 Berkeley, CA 94707 (510)524-3246

In order to minimize code size, the static method `Exception::vStandardizeExceptions()` provides a way to have a function called from within a block that will catch all errors and convert them to a subclass of `Exception`. If you plan to support other exception classes, such as the ones in the C++ standard library, you would modify this function to do the right thing.

```
OSStatus Exception::vStandardizeExceptions
(VAProc proc, va_list arg)
{
    StdException exc(GetLocationInCode());
    try
        // Call the proc
    {
        return (*proc)(arg);
    }
    catch (Exception& err)    // Exceptions are OK
    {
        throw /*err*/;
    }
    catch (char* msg)
    {
        exc.PutErrorParamPtr(
            keyErrorString, typeChar, msg, strlen(msg));
    }
    catch (long num)
    {
        exc.SetStatus(num);
    }
    catch (...)
    {
    }

    if (LogExceptions())
    {
        exc.Log();
    }

    exc.AboutToThrow();
    throw exc;
    return 0;
}
```

There are several other convenience routines, all of which call through `Exception::vStandardizeExceptions()`, which capture all exceptions and convert them to an `OSErr` or write them into an `AppleEvent`. For instance, the following can be used by an `AppleEvent` handler to catch all errors and return them in the event:

```
OSErr Exception::CatchAEErrors(AppleEvent* event,
                               VAProc proc, ...)
{
    va_list arg; va_start(arg, proc);
    OSStatus status;
    try
    {
        status = vStandardizeExceptions(proc, arg);
    }
    catch (Exception& exc)
    {
        status = exc.GetOSErr();
        if (event && event->dataHandle != nil)
        {
            if (status != errAEEEventNotHandled)
            {
                // AppleScript has an undocumented "feature"
                // where if we put an error parameter in an
                // unhandled event, it reports an error rather
                // than trying the system handlers.
                GetLocationInCode().LogIfErr(
                    exc.GetAEPParams(*event, false));
            }
        }
    }

    va_end(arg);
    if (status <= SHRT_MAX && status >= SHRT_MIN)
```

```

    {
        return (OSErr) status;
    }
    else
    {
        return eGeneralErr;
    }
}
```

This pair of functions reports all errors to the user. (The Exceptions library allows the programmer to install a callback to report exceptions to the user. Not that here we use `vStandardizeExceptions` to insure that all exceptions are converted to a subclass of `Exception()`.)

```
static OSStatus report_exception(va_list arg)
{
    VA_ARG(Exception*, exc, arg);
    exc->Report();
    return 0;
}

void Exception::ReportExceptions(VAProc proc, ...)
{
    va_list arg; va_start(arg, proc);
    try
    {
        GetLocationInCode().FailOSStatus(
            vStandardizeExceptions(proc, arg));
        va_end(arg);
    }
    catch (Exception& exc)
    {
        va_end(arg);
        try
        {
            StandardizeExceptions(report_exception, &exc);
        }
        catch (Exception& exc1)
        {
            exc1.Log();    // don't throw errors in reporting
        }
    }
}
```

CONCLUSION

Exception handling is both useful and practically required in robust code. However, C++ exceptions have a number of limitations which you must be aware of when you are developing code which uses operating system features not supported by the language. However, using the techniques described here, these limitations are not insurmountable.

BIBLIOGRAPHY

- [1] Bjarne Stroustrup, "The C++ Programming Language" (Third Edition), Addison-Wesley, 1997, ISBN 2-201-88954-4
- [2] Scott Meyers, "Effective C++" (Second Edition), Addison-Wesley, 1997, ISBN 0-201-92488-9
- [3] Scott Meyers, "More Effective C++", Addison-Wesley, 1996, ISBN 0-201-63371-X
- [4] P.J. Plauger, "The Draft Standard C++ Library", Prentis-Hall, 1995, ISBN 0-13-117003-1
- [5] James O. Coplien, "Advanced C++ Programming Styles and Idioms", Addison-Wesley, 1992, ISBN 0-201-54855-0

Thanks to Miro Jurisic, Elizabeth Rehfeld, and Brett Doehr for reviewing this article.

MT

by Andrew Downs

Writing an OS Shell in Java

Building a Facade

INTRODUCTION

This article describes the design and implementation of a Finder-like shell written in Java. Several custom classes (and their methods) are examined: these classes provide a subset of the Macintosh Finder's functionality. The example code presents some low-level Java programming techniques, including tracking mouse events, displaying images, and menu and folder display and handling. It also discusses several high-level issues, such as serialization and JAR files.

Java provides platform-dependent GUI functionality in the Abstract Windowing Toolkit (AWT) package, and platform-independent look and feel in the new Java Foundation Classes (Swing) API. In order to faithfully model the appearance and behavior of one specific platform on another, it is necessary to use a combination of existing and custom Java classes. Certain classes can inherit from the Java API classes, while overriding their appearance and functionality. This article presents techniques used to create a Java application (Facade) that provides some of the Macintosh Finder's capabilities and appearance. As a Java application, it can theoretically run on any platform supporting the Java 1.1 and JFC APIs. **Figure 1** provides an overview of the Facade desktop and its functionality. **Figure 2** shows a portion of the underlying object model. It contains a mixture of existing Java

classes (such as `Component` and `Window`) and numerous custom classes. In the diagram, the user interface classes are separated from the support classes for clarity.

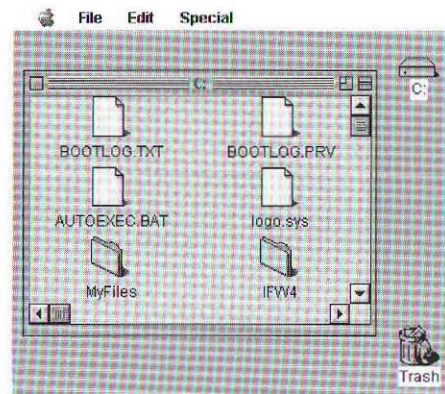


Figure 1. Facade on Windows 95.

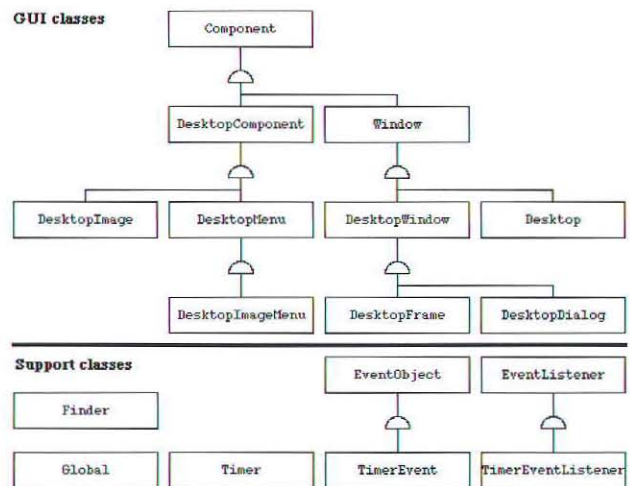


Figure 2. Facade object model (partial). Only the inheritance structure is shown; relational attributes are not shown.

Andrew Downs is a Technical Lead for Template Software in New Orleans, LA, designing and building enterprise apps. When he's not dodging hurricanes, he watches his twin sons stress test Macintosh hardware. Andrew wrote the Macintosh freeware program Recent Additions, and the Java application UDPing. You can reach him at andrew@nola.template.com.

THE OBJECT MODEL

Facade's object model utilizes the core Java packages and classes where possible. However, in order to provide the appropriate appearance and speed, custom versions of several standard Java classes were added. For instance, the `DesktopFrame` class is used to display folder contents. Its functionality is similar to the `java.awt.Frame` class. However, additional behavior (i.e. window-shade) and appearance requirements dictated the creation of a "knockoff" class.

One advantage of using custom classes can be improved speed. This holds true for classes inheriting from `java.awt.Component`: such classes do not require the creation of platform-specific "peer" objects at runtime. Less baggage results in a faster response. For example, the Facade menu display and handling classes respond very quickly.

In the custom classes, most drawing uses primitive functions to draw text, fill rectangles, etc. Layout and display customization is assisted through the use of global constant values which typically signify offsets from a particular location (as opposed to hardcoded coordinates).

Several of the classes depicted in the object model will be discussed in this article.

APPEARANCE AND FUNCTIONALITY

Facade relies on several graphical user interface classes in order to provide a Finder-like interface. These classes fall into the following categories:

- GUI
 - Desktop
 - Menus
 - Icons
 - Folders
- Support
 - Startup
 - Timing
 - Constants

Facade provides a subset of the following operations:

- Menu and menu item selection
- Displaying volume/folder contents
- Dragging icons
- Opening and closing windows
- Managing the trash
- Saving the desktop state at shutdown (exit)
- Restoring the desktop state at startup
- Launching applications
- Dialog display
- Cut/Copy/Paste
- Changing the cursor
- Emulating the desktop database

Several of these operations (and the classes which implement them) will be discussed in the following sections.

THE DESKTOP

The `Desktop` class is a direct descendant of the `java.awt.Window` class, which provides a container with no predefined border (unlike the `java.awt.Frame` class). This approach allows the drawing and positioning of elements within the Desktop area, without needing to hide the platform-specific scrollbars, title bar, etc.

However, this approach means that special care and feeding is required to make the menubar work properly. In the current implementation, menubar (and menu) drawing is done directly in the `Desktop` class, using values obtained from those respective classes. In other words, menus don't draw themselves, `Desktop` does it for them.

MENU CREATION

The following code snippet (Listing 1), taken from the `Desktop` constructor, creates the Apple menu, and populates it with one item. The menu is an instance of the `DesktopImage` class. The Apple icon (`imgApple`) was loaded further up in this method. A similar sequence creates and populates the File menu.

Listing 1: Desktop constructor

```
Creating menus in the Desktop constructor. Desktop constructor

// -----
//      • Desktop constructor
// -----

Desktop() {

    // <snip>

    // Create the menus and their menu items.
    DesktopImageMenu dim;
    Vector vectorMenuItems;
    DesktopMenuItem dmi;

    // Calculate the y-coordinate of the menus (constant).
    int newY = this.getY() + this.getMenuBarHeight() -
        ( this.getMenuBarHeight() / 3 );

    // Calculate the x-coordinate of the menus (variable).
    int newX = this.getX();

    // Create the Apple menu.
    dim = new DesktopImageMenu();
    dim.setImage( imgApple.getImage() );
    dim.setX( imgApple.getX() );
    dim.setY( imgApple.getY() );

    // Create menu item "About..."
    dmi = new DesktopMenuItem( Global.menuItemAboutMac );
    dmi.setEnabled( true );

    // Add the menu item to the Vector for this particular menu...
    dim.getVector().addElement( dmi );

    // ...then tell the menu to calculate the item position(s).
    // Note that the Desktop's FontMetrics object gets used here.
    dim.setItemLocations( this.getGraphics().getFontMetrics() );

    // Add the Apple menu to the menubar's Vector.
    // The menubar was instantiated further up in this method.
    dmb.getVector().addElement( dim );

    // For the next menu, calculate its x-coordinate.
    newX = imgApple.getX();
    newX += ( 2 * Global.defaultMenuSep );
    newX += ( ( DesktopImageMenu )dmb.getVector().elementAt( 0 )
        ).getImage().getWidth( this ); // Line wrap.
    DesktopMenu dm;

    // Create the File menu...
    dm = new DesktopMenu();
    dm.setLabel( Global.menuFile );
    dm.setX( newX );
    dm.setY( newY );

    // ...and all its menu items.
    dmi = new DesktopMenuItem( Global.menuItemNew );
```



```

dmi.setEnabled( true );
dm.getVector().addElement( dmi );
dmi = new DesktopMenuItem( Global.menuItemOpen );
dmi.setEnabled( false );
dm.getVector().addElement( dmi );
dmi = new DesktopMenuItem( Global.menuItemClose );
dmi.setEnabled( false );
dm.getVector().addElement( dmi );
//Tell the menu to calculate the item position(s).
dm.setItemLocations( this.getGraphics().getFontMetrics() );
//Add the File menu to the menubar's Vector
dmb.getVector().addElement( dm );
newX += ( 2 * Global.defaultMenuSep );
newX += theFontMetrics.stringWidth( dm.getLabel() );
//<etc.>
}

```

Here, the variable `dim` is an instance of the class `DesktopImageMenu`. This specialized menu class simply adds an instance variable that contains an `Image` (in this case, the Apple) to the `DesktopMenu` class. Its behavior is the same as other menus, except that instead of a `String` it displays its `Image`. The `x` and `y` coordinates of this menu are determined by the same values assigned to the image when it was loaded.

This object contains one menu item, the "About" item. That item is enabled, and added to the `Vector` for the menu. This `Vector` contains all the menu items for that menu. This approach provides an easy way to manage and iterate over the menu items.

Once the `Vector` has been setup, its contents are given their `x-y` coordinates for drawing and selection purposes through the `setItemLocations()` method. Although this calculation can be done when the menu is selected, the code runs faster if those numbers are calculated and assigned at startup time.

Finally, the menu is added to the `Vector` for the menubar. The menubar uses a `Vector` to iterate over its menus, in the same way the menus iterate over their menu items. This will become apparent when examining the mouse-event handling code for the `Desktop` class.

The same approach is shown for the File menu, except that File contains a `String` instead of an `Image`, as well as several menu items.

MENU AND MENU ITEM SELECTION

This class defines some global values that are shared between the `Display` and `ModeSelector` classes. These values are collected in one place to simplify housekeeping and maintenance. We will refer to them from the other classes as `Global.NORMAL`, `Global.sleep`, etc. This is the Java syntax for referencing *static* (class) attributes. Note that *final* means the values cannot be changed after they are initially assigned, so these are constants.



Figure 3. Menu item selection.

The code in Listing 2 handles `mouseDown` events in the menubar. The first two lines reset the instance variables used to track the currently active menu and menu item. Since the user has pressed the mouse, the old values do not apply. Next, the `Graphics` and corresponding `FontMetrics` objects for the `Desktop` instance are retrieved. They will be used in drawing and determining what selection the user has made.

Next, test to see whether the event occurred within the bounding rectangle of the menubar. This line uses the `java.awt.Component.contains()` method. If the `x-y` coordinates of the mouse event are inside the menubar, then determine which menu (if any) the user selected.

Determining the menu selection uses the menubar's `Vector` of menus. Iterate over the `Vector` elements, casting each retrieved element to a `DesktopMenu` object. (`Vector.elementAt()` returns `Object` instances by default, which won't work here.) The menu has its own bounding rectangle, which is compared to the event `x-y` coordinates. In addition, in order to duplicate the Finder, a fixed pixel amount is subtracted from the left-side of the bounding rectangle, and added to the right-side. This allows the user to select near, but not necessarily directly on, the menu name (or image). Notice also in the (big and nasty) if conditional that the `DesktopMenu` retrieved from the `Vector` is checked for an `Image` (this handles the Apple menu). If it has one, the image width is used instead of the `String` width.

Once the user's menu selection has been found, it is redrawn with a blue background. Then, the menu items belonging to that

THE COMPLETE LICENSE SOLUTION FOR

MAC OS X

JUST DROP IN THE...

LICENSER KIT[®]

- Integrate into your apps in minutes
- Add licenses on the fly
- Tie license to user name
- Tie to host ID
- Allow as many users as desired
- Generate expiring licenses for "full strength" demos
- Create floating network licenses
- Have unlimited number of licenses on a network
- Automatically unlicense if application is moved or copied
- Comes with sample app and license key generator
- Works on: **Mac OS X Server**, Yellow Box for Windows, OpenStep and **Mac OS X**.

INTRODUCTORY PRICE

\$1995.00

FROM YOUR FRIENDS AT

caffeine
software



www.stone.com/Licenser/ info@stone.com (505)-345-4800

menu are drawn inside a bounding rectangle (black text on white background). The menu item coordinates, and the corresponding bounding rectangle coordinates, were calculated when the menu was created, saving some CPU cycles here.

Listing 2: mousePressed

```

mousePressed

The mousePressed method handles menu selections.

// -----
// • mousePressed
// -----

public void mousePressed( MouseEvent e ) {

    // New click means no previous selection.
    this.activeMenu = -1;
    this.activeMenuItem = -1;
    Graphics g = this.getGraphics();
    FontMetrics theFontMetrics = g.getFontMetrics();
    if ( dmb.contains( e.getX(), e.getY() ) ) {
        // Handle menu selection.
        for ( int i = 0; i < dmb.getVector().size(); i++ ) {
            // Get menu object.
            DesktopMenu d = ( DesktopMenu )dmb.getVector().elementAt( i );
            // Determine if we're inside this menu.
            // This could be done with one or more Rectangles.
            // Note the (buried) conditional operator: it accounts
            // for both text and image (e.g. the Apple) menus.
            if ( ( e.getX() >= d.getX() - Global.defaultMenuHSep )
                && ( e.getX() <= ( d.getX() + ( d.getLabel() == null ?
                    ( ( DesktopImageMenu )d ).getImage().getWidth( this ) :
                    theFontMetrics.stringWidth( d.getLabel() ) ) +
                    Global.defaultMenuHSep ) )
                && ( e.getY() >= this.getY() ) && ( e.getY() <=
                    this.getMenuBarHeight() ) ) {
                // Draw menubar highlighting..
                g.setColor( Color.blue );
                // Save the current Rectangle surrounding the menu.
                // This will speed up painting on the following line,
                // and when the user leaves this menu.
                activeMenuRect = new Rectangle( d.getX() -
                    Global.defaultMenuHSep, this.getY(), ( d.getLabel() ==
                    null ? ( ( DesktopImageMenu )d ).getImage().getWidth(
                    this ) : theFontMetrics.stringWidth( d.getLabel() ) ) +
                    ( 2 * Global.defaultMenuHSep ), getMenuBarHeight() );
                g.fillRect( activeMenuRect.x, activeMenuRect.y,
                    activeMenuRect.width, activeMenuRect.height );
                // Draw menu String or Image.
                g.setColor( Color.black );
                if ( d.getLabel() != null )
                    g.drawString( d.getLabel(), d.getX(), d.getY() );
                else
                    g.drawImage( ( ( DesktopImageMenu )d ).getImage(),
                        d.getX(), d.getY(), this );

                // Get menu item vector.
                Vector v = d.getVector();
                // If the Trash is full, enable the menu item.
                // This code can easily be moved; it is included
                // here for illustration.
                DesktopMenuItem dmi =
                    ( DesktopMenuItem )v.elementAt( 0 );
                if ( dmi.getLabel().equals( Global.menuItemEmptyTrash ) ) {
                    DesktopImage di = ( ( DesktopImage
                        )this.vectorImages.elementAt( 1 ) );
                    String path = di.getPath();
                    File f = new File( path, Global.trashDisplayString );
                    if ( f.exists() ) {
                        String array[] = f.list();

                        if ( array == null || array.length == 0 )
                            dmi.setEnabled( false );
                        else
                            dmi.setEnabled( true );
                    }
                }
            }
        }
    }
}

```

```

// Draw menu background.
g.setColor( dmb.getBackground() );
g.fillRect( d.getItemBounds().getBounds().x,
    d.getItemBounds().getBounds().y,
    d.getItemBounds().getBounds().width,
    d.getItemBounds().getBounds().height );

// Draw menu items.
for ( int j = 0; j < v.size(); j++ ) {
    g.setColor( Color.black );
    if ( !( ( DesktopMenuItem )v.elementAt( j )
        ).getEnabled() )
        g.setColor( Color.lightGray );
    g.drawString( ( ( DesktopMenuItem )v.elementAt( j )
        ).getLabel(), ( ( DesktopMenuItem )v.elementAt( j )
        ).getDrawPoint().x,
        ( ( DesktopMenuItem )v.elementAt( j )
        ).getDrawPoint().y );
    g.setColor( Color.black );
    // Outline the menu item list bounding rectangle.
    g.drawRect( d.getItemBounds().getBounds().x,
        d.getItemBounds().getBounds().y,
        d.getItemBounds().getBounds().width,
        d.getItemBounds().getBounds().height );
    // Add horizontal drop shadow.
    g.fillRect( d.getItemBounds().getBounds().x + 2,
        d.getItemBounds().getBounds().y +
        d.getItemBounds().getBounds().height,
        d.getItemBounds().getBounds().width, 2 );
    // Add vertical drop shadow.
    g.fillRect( d.getItemBounds().getBounds().x +
        d.getItemBounds().getBounds().width,
        d.getItemBounds().getBounds().y + 2, 2,
        d.getItemBounds().getBounds().height );
    // Set current menu.
    this.activeMenu = i;
    // Once found, we're done.
    break;
}
}
}

```

The `mouseDragged()` method (not shown) is responsible for the highlighting and unhighlighting of menus and menu items: that is where menu items get redrawn with white text on a blue background, as depicted in **Figure 3**.

DISPLAYING FOLDER CONTENTS

Folders in Facade use a combination of core Java classes and Swing classes. The container itself descends from `java.awt.Window`, and the `paint()` method performs the low-level drawing calls (drawing the title bar, close box, etc.). The content is displayed inside a `JScrollPane`.

Once the Macintosh look-and-feel is activated, the scrollbars take the appearance shown in Figure 4. The icons within the scrollpane are added to a grid layout. This approach works well, except that the tendency of the scrollpane is to take up the entire display area (ignoring the title bar, etc.). So, on resize of the container, the scrollpane is also resized. A `java.awt.Insets` object is used to make this as painless as possible: the `Insets` values are used as the buffer area around the scrollpane.

Like most of the Facade classes, `DesktopFolder` implements the `MouseListener` and `MouseMotionListener` interfaces so it can receive mouse events. Within the methods required for those interfaces, the x-y coordinates are checked to determine if a close, zoom, shade, grow, or drag operation is taking place, and the container gets reshaped appropriately.

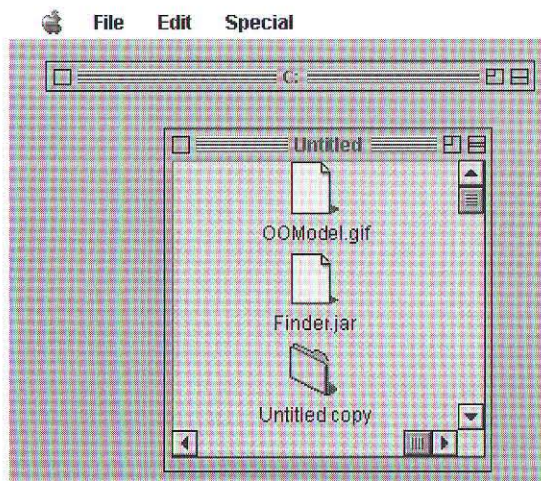


Figure 4. Displaying folder contents.

Listing 3: addFileTree

```
Building a folder's display area.                                addFileTree

// -----
// • addFileTree
// -----

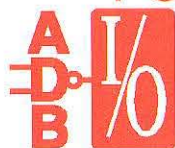
public void addFileTree( String s ) {
```

```
// Instantiate the instance variables for this object's content.
// The Mac L&F requires both scrollbars.
pane = new JScrollPane();
pane.setHorizontalScrollBarPolicy(
    JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS );
pane.setVerticalScrollBarPolicy(
    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS );
// Make sure the argument passed in is a valid directory.
String defaultVolume = s;
if ( defaultVolume == null )
    return;

File theDir = new File( defaultVolume );
if ( !theDir.isDirectory() )
    return;
// Retrieve the contents of this folder/directory.
contents = theDir.list();
int tempLength = 0;
if ( contents != null ) {
    tempLength = contents.length;
}
// Use a Swing panel inside the Swing scrollpane.
// Default to a grid layout simply because it looks the best.
JPanel p = new JPanel();
p.setLayout( new GridLayout( 5, 3, 5, 5 ) );
// If the contents contain full path specifications, there will
// be some extraneous characters that we don't want to display.
char pathSeparatorChar = theDir.pathSeparatorChar;
char separatorChar = theDir.separatorChar;
Vector v = new Vector();
v.addElement( Global.spaceSep );
int loc = 0;
int k = 0;

for ( int j = 0; j < tempLength; j++ ) {
    // For each item, separator chars should become spaces for
    // display purposes.
```

Mac, go out and touch the world for only \$199!



ADB I/O lets your customers' Macs control things, it lets them feel.
ADB I/O lets the Mac be part of the physical world.

Thousands of Uses

Science, Multimedia, Children's Museums, Home Automation,
Theatre Stages, Industrial Testing, Medical Research, Bonsai
Watering, Robotics, Weather Stations—anything that can

be electronically measured or controlled can use the ADB I/O.

No Serial Ports Occupied

ADB I/O uses the Apple Desktop Bus to communicate inputs and
outputs to and from your Macintosh. (Maximum polling frequency
is 90 Hz.) No external power supply is needed.

Eight I/O Channels Provided

Four relays for output. Four channels for Digital In,
Digital Out or 8-bit Analog In.

Extensive Software Support

With ADB I/O and nearly any environment,*
it is easy to build customized
applications for your control and
data acquisition needs.

For more info, visit us at
www.bzzzzzz.com.



*ADB I/O supports three language environments, two scripting environments, multimedia development environments, and other specific application environments with more on the way. Visit our home page to find out more and to download the complete manual as well as all supporting software at: <http://www.bzzzzzz.com>.
(818) 304-0664 voice. e-mail: contact@mail.bzzzzzz.com (818) 568-1530 fax. © 1997 Beehive Technologies, Inc. All rights reserved.


```

File tempFile = new File( theDir, contents[ j ] );
contents[ j ] =
new String( contents[ j ].replace( pathSeparatorChar, ' ' ) );
contents[ j ] =
new String( contents[ j ].replace( separatorChar, ' ' ) );
for ( int l = 0; l < v.size(); l++ ) {
// Parse root volume name.
// Remove leading "%20" character.
loc = contents[ j ].indexOf( ( String )v.elementAt( l ) );
if ( loc == 0 )
contents[ j ] = new String( contents[ j ].substring( (
( String )v.elementAt( l ) ).length() ) );
//Volume name includes first "%20".
// Rework this for MacOS.
loc = contents[ j ].indexOf( ( String )v.elementAt( l ) );
// Build the final display String from substrings.
while ( ( loc > 0 ) && ( loc < contents[ j ].length() + 1 ) ) {
String s1 = new String( contents[ j ].substring( 0, loc ) );
String s2 = new String( contents[ j ].substring( loc + (
( String )v.elementAt( l ) ).length() ) );
contents[ j ] = new String( s1 + " " + s2 );
loc = contents[ j ].indexOf( ( String )v.elementAt( l ) );
}
}

// Now build the appropriate icon.
Image theImage;
int tempWidth = 0;
DesktopFrameItem d;
ImageIcon ii = new ImageIcon();
// Files obviously look different than folders.
// Set the appropriate Image.
// This version does not handle mouse clicks on documents.
if ( tempFile.isFile() ) {
d = new DesktopDoc();
ii = new ImageIcon( ( ( DesktopDoc )d ).getImage() );
}
else {
d = new DesktopFolder();
ii = new ImageIcon( ( ( DesktopFolder )d ).getImage() );
d.addMouseListener( ( DesktopFolder )d );
}

// Set the display Strings.
d.setLabel( contents[ j ] );
d.setText( contents[ j ] );
// Set the path for the item. It is built from the parent folder
// path and filename.
d.setPath( this.getPath() +
System.getProperty( "file.separator" ) + contents[ j ] );
// And set the icon.
d.setIcon( ii );
// Swing methods for positioning the icon and label.
d.setHorizontalAlignment( JLabel.CENTER );
d.setHorizontalTextPosition( JLabel.CENTER );
d.setVerticalTextPosition( JLabel.BOTTOM );
ii.setImageObserver( d );
// Add the completed item to the panel.
p.add( d );
}

// Set the panel characteristics, then add it to the scrollpane.
p.setVisible( true );
pane.getViewPort().add( p, "Center" );
pane.setVisible( true );
// A container within a container within a container...it works.
panel = new JPanel();
panel.setLayout( new BorderLayout() );
panel.add( pane, "Center" );
this.add( panel );
// Use the Insets object for this window to set the viewing
// size of the panels and scrollpane.
panel.reshape( this.getInsets().left, this.getInsets().top,
this.getBounds().width - this.getInsets().right -
this.getInsets().left, this.getBounds().height -
this.getInsets().top - this.getInsets().bottom );
// Ready for display.
panel.setVisible( true );
this.pack();
this.validate();
}

```

THE TRASH

The trash is simply a directory located at the root of the volume (for example, `c:\Trash`). Items can be dragged onto the trash can icon, the mouse button released, and the item (and its contents, if it is a folder) are moved to the trash directory. If the item has an open window, that window is destroyed. Emptying is accomplished through the "Empty Trash..." menu item in the Special menu. **Figure 4** shows a folder being dragged to the trash. The cursor did not get captured in the image, but it is positioned over the trash icon, ready to release the folder.

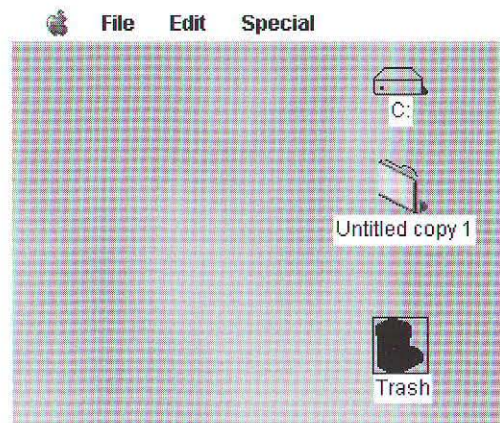


Figure 5. Dragging a folder to the Trash.

The code in Listing 4 handles the "Empty Trash..." menu item. If a match is found between the selected item and the constant assigned as the Trash label, the path to the Trash is retrieved. Once the code determines the "Trash" is indeed an existing directory, the `emptyDir()` method is called. Once `emptyDir()` returns, the icon is changed from full to empty. Note that the trash icon is always present in the `vectorImages` object. This `Vector` contains the known images (icons) for items on the desktop. The trash is always at position 0, the first position. The root volume is at position 1. This allows quick, though hardcoded, access to the images when repainting occurs, or an image needs swapping.

Listing 4: handleMenuItem

```

handleMenuItem
Handling the "Empty Trash..." menu item.

// -----
// * handleMenuItem
// -----

public boolean handleMenuItem() {
    boolean returnValue = false;
    // Handle active menu item.
    if ( ( this.activeMenu != -1 ) && ( this.activeMenuItem != -1 ) ) {
        // Get menu object.
        DesktopMenu dm =
        ( DesktopMenu )dmb.getVector().elementAt( this.activeMenu );
        // Get menu item.
        String s = ( ( DesktopMenuItem )dm.getVector().elementAt(
        this.activeMenuItem ) ).getLabel();
        // <snip>
        if ( s.equals( Global.menuItemEmptyTrash ) ) {
            // The path to the trash is assumed to be of the form:

```



```
// <root vol>/Trash
DesktopImage di =
    ( ( DesktopImage )this.vectorImages.elementAt( 1 ) );
String path = di.getPath();
File f = new File( path, Global.trashDisplayString );
// Once we have the object that references the Trash dir...
if ( f.exists() && f.isDirectory() ) {
// Call the method which empties it.
boolean b = this.emptyDir( f );
// If successful, change the icon back to empty.
if ( b ) {
// Save the current bounds. We really want the x-y.
Rectangle r =
    ( ( DesktopImage )this.vectorImages.elementAt( 0 )
    ).getBounds();
// Change the icon. imgTrash and imgTrashFull are two
// instance variables, each referencing the appropriate
// icon.
this.vectorImages.setElementAt( this.imgTrash, 0 );
( ( DesktopImage )this.vectorImages.elementAt( 0 )
    ).setBounds( r );
// Show the change.
this.repaint();
}
}
}
```

EMPTYING THE TRASH

`emptyDir()` is displayed in Listing 5. This method will empty the specified directory recursively. As it finds each item in the directory, appropriate action is taken. If the item is a file it is immediately deleted. If it is a directory, `emptyDir()` is called again, with the subdirectory name as its argument. Eventually, all of the files in the

subdirectory are deleted, and then the subdirectory itself is removed.

Listing 5: emptyDir

```
Emptying a directory.
emptyDir

// -----
// * emptyDir
// -----

public boolean emptyDir( File dir ) {
// Recursive method to remove a directory's contents,
// then delete the directory.
boolean b = false;
// Get the directory contents.
String array[] = dir.list();
// If the path is screwed up, this will catch it.
if ( array != null ) {
// Iterate over the directory contents...
for ( int count = 0; count < array.length; count ++ ) {
    String temp = array[ count ];
// Create a new File object using path + filename.
File fl = new File( dir, temp );
// Delete files immediately.
// Call this method again for subdirectories.
if ( fl.isFile() ) {
    b = fl.delete();
}
else if ( fl.isDirectory() ) {
    b = this.emptyDir( fl );
    b = fl.delete();
}
}
}
return b;
}
```

Write once, run away.

All right, you might not be able to just pack up and head straight to the beach. But the crossplatform, cross-language, and across-the-network compatibility of NeoLogic™'s embedded object database engine will save you so much time, you can at least start shopping for sunblock.

After all, with full binary compatibility across all platforms, NeoAccess® lets you create a database on one platform, and then simply open it on another, without conversion. And, because the API is exactly the same on each platform, you can code in the development environment you know best and run your application anywhere.

The result? Users of your product will be able to move around freely—across machines, languages, and platforms. Which is precisely why companies like Netscape® and NetObjects® use our technology, and why you should too.



With a paltry memory fingerprint and a 10x performance advantage over competing products, NeoAccess lets you use system resources in ways that make the most sense for your application. You can also support multiple transactions and users with our NeoShare® client/server database engine, which—like NeoAccess—is Microsoft COM®-compatible.

Not only that, if you need to use C++ together with other interfaces, our NeoOpen® product provides support for standards like ODBC.

Want more information? Run on over to www.neologic.com for a closer look.



neo•logic

NeoLogic Systems, Inc. • Tel. 510-524-5897 • Fax. 510-524-4501
www.neologic.com • neologic@neologic.com

SAVING AND RESTORING STATE

Facade uses a variant on the standard Java serialization mechanism for saving the Desktop state. The code in Listing 6 illustrates the saving and restoring of open folder windows. Both of these methods are in the Desktop class. Note that rather than flatten entire DesktopFrame objects, this code saves only specific attributes from each object. The primary reason for this approach is that it avoids any exceptions thrown when attempting to serialize the Swing components and Images (which are not serializable by default) within each DesktopFrame. Another reason is the relative ease with which this approach may be implemented. Plus, it reduces the amount of data written to disk. One disadvantage is that the current scroll position of any open window is lost.

Listing 6: saveState and restoreState

```

Saving and restoring open DesktopFrames.
saveState

// -----
//      * saveState
// -----

private void saveState() {
    //The Desktop data will be stored in a file at the root level.
    DesktopImage di =
        ( ( DesktopImage )this.vectorImages.elementAt( 1 ) );
    String s = new String( di.getPath() + "Desktop.ser" );
    try {
        FileOutputStream fos = new FileOutputStream( s );
        ObjectOutputStream outStream = new ObjectOutputStream( fos );
        // Use a temporary Vector to hold the open DesktopFrames.
        // This should not be necessary when shutting down, but it
        // is a good habit to follow.
        Vector v = new Vector();
        v = ( Vector )this.vectorWindows.clone();
        // Use another temporary Vector to hold the attributes to save.
        Vector temp = new Vector();

        for ( int i = 0; i < v.size(); i++ ) {
            // For each DesktopFrame, we'll save its path, display
            // string, and its bounding Rectangle. If window-shading is
            // in effect on an object, restore the full height before
            // saving.
            DesktopFrame df = ( DesktopFrame )v.elementAt( i );
            temp.addElement( df.getPath() );
            temp.addElement( df.getLabel() );
            if ( df.getShade() ) {
                df.restoreHeight();
            }
            temp.addElement( df.getBounds() );
        }
        // Write the Vector contents to the .ser file.
        outStream.writeObject( temp );
        outStream.flush();
        outStream.close();
    }
    catch ( IOException e ) {
        System.out.println( "Couldn't save state." );
        System.out.println( "s = " + s );
        e.printStackTrace();
    }
}

// -----
//      * restoreState
// -----

private void restoreState() {
    //The Desktop data is stored in a file at the root level.
    //This step occurs near the end of the Desktop constructor,
    //and so can use the root volume to build the path.
    DesktopImage di =
        ( ( DesktopImage )this.vectorImages.elementAt( 1 ) );

```

```

String s = new String( "\\Desktop.ser" );
try {
    // Open the file, and read the contents. In this version
    // we know it's just one Vector.
    FileInputStream fis = new FileInputStream( s );
    ObjectInputStream inStream = new ObjectInputStream( fis );
    Vector v = new Vector();
    v = ( Vector )inStream.readObject();
    inStream.close();
    Rectangle r = new Rectangle();
    for ( int i = 0; i < v.size(); i++ ) {
        // Iterate over the Vector contents. We know the save format:
        // each field corresponds to a specific attribute of a
        // DesktopFrame.
        // Create a new DesktopFrame using the retrieved path.
        s = ( String )v.elementAt( i );
        DesktopFrame df = new DesktopFrame( s );

        // Set its label using the next Vector element.
        i++;
        s = ( String )v.elementAt( i );
        df.setLabel( s );
        // Set its bounding Rectangle using the next Vector element.
        i++;
        r = ( Rectangle )v.elementAt( i );
        df.setBounds( r );
        // Prepare and display the DesktopFrame.
        df.pack();
        df.validate();
        df.show();
        df.toFront();
        df.repaint();
        // Add the object to the Desktop's Vector of open windows.
        Desktop.addWindow( df );
    }
    catch ( ClassNotFoundException e ) {
        System.out.println( "ClassNotFoundException in
restoreState()." );
    }
    catch ( IOException e ) {
        System.out.println( "Couldn't retrieve state." );
        System.out.println( "s = " + s );
        e.printStackTrace();
    }
}

```

PACKAGING (FACADE IN A JAR)

Facade can be run from a Java Archive (JAR) file, or as a set of separate classes plus images. In addition, the Macintosh look-and-feel classes must be present (usually in a separate JAR). The system environment variables need to be setup properly in order for the Java runtime to find the classes.

The JAR file for Facade was created like this:

```
jar cf Facade.jar /Facade/*.class /Facade/*.gif
```

Once the environment variables are set, Facade can be invoked as follows:

```
java Facade
```

OTHER CLASSES

In addition to the `java.awt` classes already discussed, there are other packages and classes used often in Facade.

This program relies heavily on the `java.util.Vector` class for maintaining a semblance of order among the various objects. Several `Vector` instances are used for tracking icons and open windows. **Figure 6** illustrates the core concept of a `Vector`: it is a growable array.

NEW!

VOODOO Server - Version Control the Macintosh Way

VOODOO Server

The successor of our award-winning VOODOO. More powerful and still easy to use - just as Macintosh software should be...



- ◆ Efficient, robust and easy-to-use version control for CodeWarrior developers.
- ◆ A powerful, solid concept with a clear, Mac-like user interface, all integrated in the CodeWarrior IDE.
- ◆ Client/server architecture provides a robust and efficient solution, scalable from single developers to large groups.
- ◆ Improved delta compression saves up to 99 % of disk space for arbitrary files.
- ◆ Fully scriptable - based on the AppleEvent Object Model.

... just use it and forget the bureaucratic overhead of other version control tools.

◆ Download a fully functional trial version from our web site.

◆ Visit our booth at MacWorld Expo San Francisco (Developer Central).

UNI SOFTWARE PLUS, A-4232 Hagenberg, Austria
email: voodoo@unisoft.co.at, <http://www.unisoft.co.at/products/voodooserver.html>

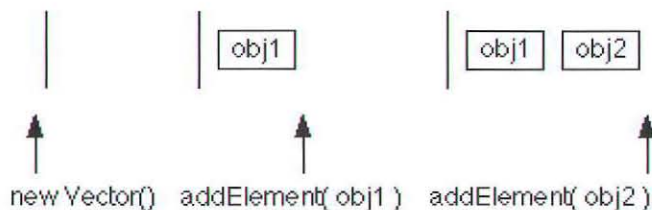


Figure 6. Creating and populating a Vector. The arrow represents a pointer to the current element at the end of the Vector.

The `java.io.File` class provides most of the file system operations for display and management. Refer to the references provided in the bibliography for API details. One platform-specific detail that was not discussed is the translation of a root path to a form that is valid for a `File` directory object. For example, calling the `File.list()` method on the path "c:\" will not return any file or folder names, but calling the same method for the path "/" will return the root directory contents.

Facade also uses the Java Foundation Classes (Swing) to provide a consistent look and feel across platforms, primarily for window scrolling operations. Swing runs a bit slow right now, but as performance improves Swing classes can be substituted for some of the Facade custom classes. For example, the code that handles folder content display can be modified to show a tree structure. This capability should be easy to implement using the Swing classes, with little additional low-level drawing code.

CONCLUSION

Writing Java classes to model operating system functionality requires some choices, particularly in the selection of pre-defined GUI classes. Although many classes are available, you will need to create others, since performance of the pre-defined classes may be slow, or the behavior may not be exactly what you need. Both the appearance and behavior for custom GUI elements must be written.

This article touched on several key areas: desktop layout, menu and mouse handling, icon display and movement, the trash, and saving and restoring the desktop state. All of these functional areas can be modeled easily using the Java 1.1 classes, supplemented by JFC and custom-built classes.

REFERENCES

- Developing Java Beans, Robert Englander, O'Reilly & Associates, Inc., 1997.
- Java in a Nutshell, David Flanagan, O'Reilly & Associates, Inc., 1997.
- Exploring Java, Patrick Niemeyer and Joshua Peck, O'Reilly & Associates, Inc., 1997.
- Programming with JFC, Scott Weiner and Stephen Asbury, John Wiley & Sons, Inc., New York, NY, 1998.

URLs

- <http://developer.apple.com/java/>
- <http://www.lists.apple.com/mrj.html>
- <http://java.sun.com/products/index.html>

MT

No Cute See-Through Green Panels

Unlike some high-tech companies, Zocalo isn't reaching out to the mass market. We've been serving business, industry, and higher education exclusively for more than a decade. So your business' Internet traffic doesn't queue up behind some kid's game of Doom, and your network staff don't have to wait on hold to speak to our technicians.

By focusing on the high-speed networking needs of businesses, we've been able to build an Internet backbone that provides better than 99.995% overall uptime. With peering at more West Coast locations than any other ISP, we're uniquely able to route your company's critical data around Internet slow-downs and trouble-spots. Our managed multiprotocol network is the only one of its kind, allowing Zocalo alone among

ISPs to offer native routing of AppleTalk, IPX and DECnet between customers' office LANs throughout the world, at speeds up to 45 megabits per second. We also deliver fully-configured equipment, proactive customer service, on-site maintenance, and a firewall custom-built to your specs, all at no extra charge. We take responsibility for your wide-area network and Internet connection right up to the FDDI or Ethernet jack in each of your offices.

If cute just isn't enough for your business, call our network engineers and find out why Zocalo is the first choice for industrial-strength business networking.

+1 510 540 8000
info@zocalo.net
<http://www.zocalo.net>

Multipoint Internet Access

**Z
O
C
A
L
O**

by Jessica Courtney <press_releases@mactech.com>

VOODOO SERVER

UNI SOFTWARE PLUS introduced VOODOO Server, a successor of their version control tool VOODOO, which won a MacWorld/MacTech Eddy Award last year.

VOODOO Server is a real client/server solution. This forms the basis for a highly efficient and robust multi-user environment. Efficient, because all time consuming tasks can be delegated to the server and the developer's machine is no longer slowed down (e.g., from the client's point of view, a check-in operation is reduced to the transfer of the file to the server). Robust, because the server application is the only process that works on the database. In combination with a solid transaction mechanism the client/server architecture minimizes the risk of database corruption. Since the server application can run either on a server machine anywhere on the network but also on the local machine, VOODOO Server is scalable from single user projects up to large projects with many developers.

VOODOO Server in combination with the CodeWarrior plug-in offers a user-friendly version control solution that enables you to manage your software project without having to deal with additional complexity and bureaucracy. The design goal was to give you access to all necessary information and functions without having to learn version control vocabulary and remember revision numbers like 1.4.3.5.6.2. Instead the CodeWarrior plug-in for VOODOO Server offers all the functionality and version information directly inside the CodeWarrior IDE in an easy-to-use Mac-like user interface. The package includes a tutorial that lets you learn anything you have to know within only a few hours.

VOODOO Server introduces a new concept called "Project Parts" or simply "Parts". A part represents a subtree of a project with all included files and folders. A VOODOO project consists of an arbitrary number of parts. Parts can be shared among projects. If you make changes to a part from one project, these changes are propagated to all other projects sharing this part.

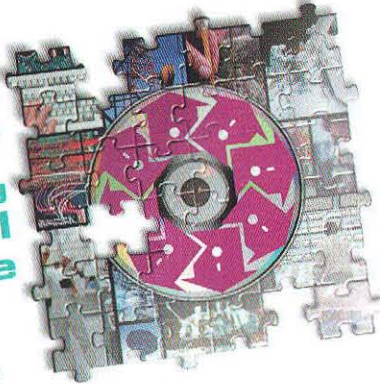
VOODOO Server keeps track not only of the different versions of the files, but also of the folder structures. This allows you to reconstruct earlier configurations of your project exactly as they appeared at that time, including the folder structure. VOODOO Server keeps also track of the renaming of files and will even realize if you delete a file from the VOODOO project and add the same file again later. <<http://www.unisoft.co.at/products/voodoooserver.html>>



Fast Mac ISAM Access

Puzzled by
database
performance
issues?

Nothing
beats... ISAM
performance



Real-world data management solutions are typically more complex when one examines the pieces, than initially recognized by the majority of database programmers. All software projects are complex puzzles comprised of many details, most of which are data-related. Often today's "DBMS" solutions sacrifice the speed or control essential for a competitive application.

c-tree Plus®, by FairCom, has been the choice of commercial developers for twenty years precisely because it offers the flexibility and control at the detail level to fit a

wide variety of data management needs. Proven on large Unix servers and workstations, c-tree Plus's small footprint and exceptional performance have also made it the engine of choice for professional developers on Mac and Windows. c-tree Plus offers sophisticated ISAM level control with which the developer may define precise data management solutions, making it a perfect fit for any development project requiring specific data handling features.

c-tree Plus® offers the most mature ISAM solution today...

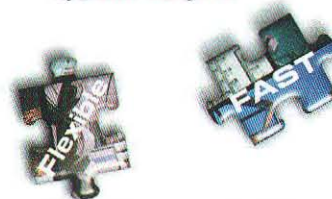
FairCom's c-tree Plus database engine:

- Advanced Indexing Technology
- Complete Source Code
- Complete Transaction Processing
- ODBC Interface from Windows clients
- Over 25 Developer's Servers Included
- Royalty Free
- Standalone, Multi-user or Client/Server Models
- Y2K Compliant
- Supports Metrowerks, Symantec Compilers

The FairCom Server:

A solid, high performance database server that is scalable, portable and offers unequalled control. FairCom has been providing database solutions to the commercial development community for twenty years and supporting Mac for nearly as long. You won't find a better Mac DBMS solution, with these features and performance anywhere else!

- Client Side Source Code
- File Encryption
- File Mirroring Logic
- Full Conditional Index Support
- Full Heterogeneous Networking
- Multiple Communication Protocols: ADSP; SPX; TCP/IP
- Online Data Backup
- Small Memory Footprint
- Flexible OEM Licensing Options
- Source Code Availability



All these platforms supported in one package:

Mac, MIPS ABI, DEC Alpha, Sun SPARC, Windows 9X, SCO, 88OPEN, AIX, RS/6000, HP9000, Sun OS, Interactive Unix, Linux (Alpha...), AT&T System V, QNX, Free BSD, OS/2, Windows NT, Windows 3.1, DOS, Netware NLM, & Banyan VINES.



Don't wait, see for yourself!
USA. 800.234.8180

www.faircom.com
Email: info@faircom.com

FairCom®

Database SOLUTIONS Since 1979

Phone: USA 573.445.6833 • EUROPE +39.035.773.464
JAPAN +81.59.229.7504 • BRAZIL +55.11.3872.9802

Other company, product and operating platform names are registered trademarks or trademarks of their respective owners.

by Jeff Clites <online@mactech.com>

Stocking Your Toolbox

The Toolbox was one of the early strengths of the Macintosh platform. From the beginning, it has armed programmers with an assortment of routines to make their job easier. Even so, there are those times when you wish the Toolbox went just a little further, or helped you out in just one more area. This month we are going to look at some third-party goodies which fill in the gaps.

TEXT

TextEdit, the Mac's built-in text-handling package, is great for simple text-handling, but everyone is aware of its limitations — the most important one being that it can't handle more than 32K of text. Fortunately for all of us, Marco Pivonelli stepped up and created WASTE, the WorldScript-Aware Styled Text Engine. It can handle larger stretches of text, limited only by available memory, but its enhancements don't stop there. As its name implies, it is WorldScript savvy; it also supports tab stops, drag-and-drop, and embedded objects. It comes with an instructive demo application and Inside Macintosh-style documentation, and it is the basis of the author's Style text editor, which shows off its features nicely. The WASTE API is modeled closely after TextEdit itself, and accordingly the programmer needs to be familiar with the standard Mac text editing routines in order to get the most out of WASTE. For those who want a direct comparison, the best place to look is in the source code for the SIOUX console window, which is part of Metrowerks' CodeWarrior. It has both a TextEdit and a WASTE implementation together in one file, making it easy to see how they correspond.

The WASTE Page

<<http://www.boingo.com/waste/>>

Timothy Paustian, author of the CWASTEEdit PowerPlant wrapper class for WASTE, has written a text engine of his own, the WTextEngine (or WT++). It is still under development, and is entirely C++ based, so it's handy for those who prefer a completely object-oriented alternative. The package includes an example application which

demonstrates adding multiple undo, drag-and-drop, and AppleScript support to the engine.

WTextEngine

<<http://www.bact.wisc.edu/WTextEngine/Overview.html>>

CWASTEEdit

<<http://www.bact.wisc.edu/CWASTEEdit/CWasteEdit.html>>

CONTROLS

I first ran across Kyle Hammond's web pages on the trail of something known as the A List. The A List is Kyle's replacement for the Toolbox's List Manager. Unlike the List Manager, it supports cut, copy, paste, and drag-and-drop, and very large lists (for those hopefully rare occasions where it is appropriate to display a huge list using this sort of UI element). Also, since it's not an LDEF it ends up being much faster than the List Manager (which still contains emulated code), and it's more flexible. For instance, it supports the use of callbacks for drawing, highlighting, and disposing of cell data, so it is easy to supply your own routines to display lists of non-text data.

Kyle's site also has a few more handy pieces of code to help you with your interface. First there is GetMultiple, a function which uses CustomGetFile to allow users to select more than one file at a time, with an interface modeled after CodeWarrior's "Add Multiple..." command. This may be partially superseded by Navigation Services, but the usual problems of backward compatibility will render it useful for a while to come. A second set of functions, EditTextCntlExtras, help with the handling of Appearance Manager editable text controls, and there are a few other interesting things there as well. It's definitely a site worth checking out. (And generously, Kyle has made all of this code available for free.)

Kyle Hammond's Mac Programming Page

<<http://genbiol.cbs.umn.edu/staff/hammond/MacProgramming.html>>

THE UNIVERSAL PPP API

It is annoyingly involved to monitor and change the state of a PPP connection, because each PPP driver (OT/PPP, FreePPP, etc.) has its own interface to do this. Luckily for us, the details have been sweated out for us by Sailmaker Software. They have written the Universal PPP API, which implements a wrapper around the various PPP interfaces, so you don't have to worry about which PPP software your user is running, even if they change it without restarting your application. It is shareware, and is distributed as a binary library (callable from C, C++ and Pascal); the source code is available for an additional charge.

The Universal PPP API

<http://www.sailmaker.co.uk/uppp_api.html>

MOREFILES

The MoreFiles distribution is such an essential collection of routines that you have to wonder why it isn't just rolled into the Mac OS itself. MoreFiles is written by Jim Luther, and grew out of his experiences working for Apple Developer Technical Support. It contains routines for doing just about anything you might want to do with files on the Mac, but which the OS doesn't already do for you — things such as getting the full path to a file (which you should *really really*

avoid doing except for display purposes) or iterating through a folder hierarchy. If you need to do something with the file system and you don't know how to do it, *look here first*. You will either see that it has already been done for you, or you'll find some useful sample code which does something similar. It is available at most Macintosh archives, but its home is on Jim Luther's curiously ftp-based web page.

Jim Luther

<<ftp://members.aol.com/JumLong/index.html>>

These and rivers of other links are available from the MacTech Online web pages at <<http://www.mactech.com/online/>>.

MT

Want to know what products
are available for Mac OS
development? Check out
Developer Depot®
<<http://www.devdepot.com>>

Rapid application
development in C++?

Get **REAL**



REALbasic

Visit www.realbasic.com for a FREE 30 day trial version
or call (512) 292-9988 for more information

REALbasic is a trademark of REAL Software, Inc.

Modern Object-Oriented BASIC
Full native compilation
PowerPC Shared Library Support
Imports VB Forms and Code
Appearance Manager Savvy
XCMD, AppleEvents and AppleScript Support
Built-in Sprite Animation Engine
...Windows and Java Compilers on the way!

**THE
CLASSIFIEDS**

Hit your target the first time!

DEPOT
www.devdepot.com

**Your one stop shop for all
your developer needs!**

PO Box 5200 • Westlake Village, CA • 91359-5200
Voice: 800/MACDEV-1 (800/622-3381) • Outside US/Canada: 805/494-9797
Fax: 805/494-9798 • E-mail: orders@devdepot.com

<http://www.scientific.com>

Professional Software Developers

Looking for career opportunities?

Check out our website!

Nationwide Service

Employment Assistance

Resume Help

Marketability Assessment

Never a fee

Scientific Placement, Inc.

800-231-5920 800-757-9003 (Fax)

das@scientific.com

Information you would **KILL** for

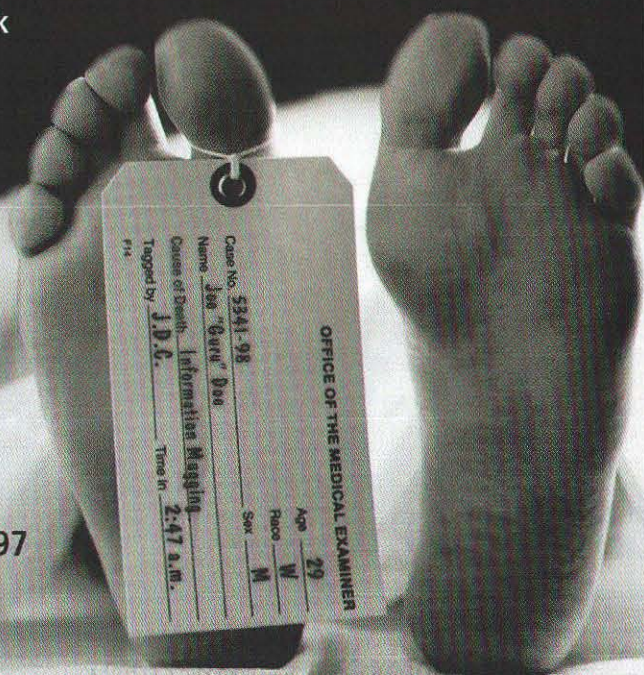
No need to kill for the information — call on the professionals. NetProfessional Magazine is where web developers and network administrators look for answers. Each issue provides you with practical and detailed solutions for real world problems, extensive how-to articles, in-depth product reviews, practical tutorials, tips, techniques, reliable technology summaries, and more!

Try a RISK FREE subscription and see for yourself — at the low price of only \$19.95 for six issues.

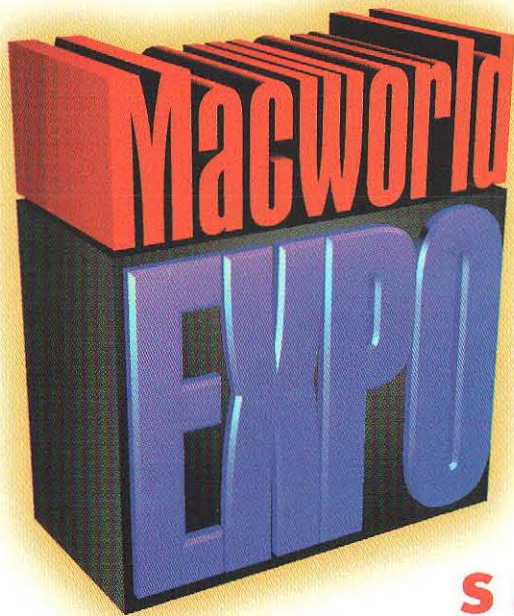


Phone: 800/622-3381
Outside US/Canada: 805/494-9797
Fax: 805/494-9798
E-mail: orders@netprolive.com

www.netprolive.com



THE Future



HERE AND NOW!

MACWORLD Expo/San Francisco

EXHIBITS: January 5-8, 1999

WORKSHOPS: January 4, 1999

MACWORLD/PRO CONFERENCE:
January 5-7, 1999

MACWORLD USERS CONFERENCE:
January 6-8, 1999

Moscone Convention Center

San Francisco, CA

For millions of computer users, Macintosh is synonymous with increased productivity and innovative technology advances. Come to San Francisco—home to a “think different” community—for the latest technologies and trends that will change the way you work, play and learn now and in the future.

At MACWORLD Expo/San Francisco, you'll demo, touch, see and feel all the hot new products—from more than 400 leading vendors. Learn time-saving productivity secrets through in-depth workshops, advanced and user-level conference programs. Hear from the brightest stars who are shaping the future of the Mac universe at the much-anticipated keynote address.

MACWORLD Expo/San Francisco has the latest solutions for:

- *the Internet*
- *digital content creation, management and delivery*
- *multimedia*
- *software development*
- *small business*
- *remote worker programs*
- *connectivity*
- *graphic design*
- *publishing*
- *education, research and development*

Take advantage of money-saving specials when you buy products right on the show floor.

**Discover what the future holds at
MACWORLD Expo/San Francisco.
Register to attend today!**

VISIT: www.macworldexpo.com
CALL: **800-645-EXPO**

**YES! PLEASE SEND ME MORE INFORMATION ABOUT
MACWORLD EXPO/SAN FRANCISCO! I'M INTERESTED IN:**

☐ **Attending**

☐ **Exhibiting**

MT

Name _____

Title _____

Company _____

Address _____

City/State/Zip _____

Phone _____ Fax _____

email _____

(if you would like to receive information via email about MACWORLD Expo)

Mail to: MACWORLD Expo, 1400 Providence Highway,
P.O. Box 9127, Norwood, MA 02062. Or Fax to: 781-440-0357

THIS IS NOT A REGISTRATION FORM.

Owned & Produced by:

WORLD EXPO

Sponsored by:
Macworld

Managed by:

EXPO MANAGEMENT
COMPANY

by Jeff Clites <online@mactech.com>

AUTOCOMPLETION IN MPW

One very useful feature of many Unix shells is that you only need to type the beginning of a command, hit tab, and it fills in the complete command. I was really missing this on MPW, especially with long command names such as UnobsoleteNameRevisions.

So here's a script that attempts to provide this feature in MPW, as well as possible. Bind it to Command-Tab (or whatever you want), then type "nam", hit Command-Tab, and watch it fill in "NameRevisions" for you.

Note: if nothing is selected, it will try to complete the last word on the current line. If you want to complete a word in the middle of a line, you'll have to select it.

If ":" is in your {Commands} variable, it will also consider all files in the current directory, as I have found no way to figure out whether a file is an MPW script or not. Actually, this may even be useful.

```
# get the current selection
Set CurrentSelection ``Catenate "{Active}".$`

If "{CurrentSelection}" == ""
    # select the entire line we're on
    Find !0 "{Active}"
    # put the insertion point at the beginning of this line
    Find Δ$:Δ$ "{Active}"
    # remember line number, so we can restore it later
    Set CurrentLine `Position -1 "{Active}"`

    # select the last word on this line
    Find /[↵ @t@n]+[ @t]*∞/ "{Active}" || (Beep; Exit 0)

    # if it isn't really on our current line, exit
    If `Position -1 "{Active}"` != {CurrentLine}
        Find {CurrentLine} "{Active}"
        Beep
        Exit 0
    End

    Set CurrentSelection ``Catenate "{Active}".$`
    # strip spaces from the end, if any
    If "{CurrentSelection}" ∼ /[↵ @t]+)@1[ @t]*/
        Set CurrentSelection "{@1}"
    End

    # can this happen at this point?
    If "{CurrentSelection}" == ""
        Beep
        Exit 0
    End
End

# a list of matching command names
Set CommandList ""
# the number of matches found
Set Matches 0

# first see if it's one of the built-in commands
# (we really have to list them all here, sigh...)
For i In @
    AddMenu AddPane Adjust Alert Alias Align AuthorInfo @
    Beep Begin Break Browser Catenate CheckIn CheckOut @
    CheckOutDir Clear Close Confirm Continue Copy Cut Date @
    Delete DeleteMenu DeleteNames DeletePane @
    DeleteRevisions Directory Duplicate @
    DuplicateNameRevisions Echo Eject Equal Erase Evaluate @
    Execute Exists Exit Export Files Find Flush @
    FlushAllVolumes For Format Help HideWindows If @
    LockNameRevisions Loop Mark Markers ModifyReadOnly @
    Monitors Mount MountProject Move MoveWindow @
    NameRevisions New Newer NewFolder NewProject @
    ObsoleteNameRevisions ObsoleteProjectorFile Open @
    Parameters Paste PlaySound Position Project ProjectInfo @
    Quit Quote Redo Rename RenameProjectorFile Replace @
    Request ResolveAlias Revert RotatePanels RotateWindows @
    RProj RShell Save SaveOnClose Set SetFile SetKey Shift @
    ShowSelection ShowWindows Shutdown SizeWindow @
    StackWindows Target TickCount TileWindows Unalias Undo @
    Unexport UnlockNameRevisions Unmark Unmount @
    UnmountProject UnobsoleteNameRevisions @
    UnobsoleteProjectorFile Unset UnsetKey @
    UpdateProjectorDatabase Version Volumes Which Windows @
    ZoomWindow
End
End

# now process every folder in the current {Commands} list
For Folder in `Which`
    For File In `Files "{Folder}" -s`

        # append all matching names to our list of command names
        If "{File}" ∼ /[CurrentSelection]∞/
            Set CommandList "{ {CommandList} } '{File}'"
            Evaluate Matches += 1
        End
    End
End

# no matching names found
If {Matches} == 0
    Find /∞/ "{Active}"
    Exit 0
End

# if more than one name was found, present a list
# to choose from
If {Matches} > 1
    Set CommandList ``GetListItem -sort -s -c {CommandList}``
    Exit 0 If "{CommandList}" == ""
Else If "{CommandList}" ∼ / (∞)@1/

    # if only one name was found, it has a space at the
    # beginning; strip it. And remove any unnecessary
    # quoting.
    Set CommandList ``Quote {@1}``
End

# replace the selection with the found (or chosen) name
Echo -n "{CommandList} " > "{Active}".$`
Find $Δ "{Active}"
```

Stefan Haller
<stk@snafu.de>



How much do you pay for network consultation?

How does FREE sound?



Get Your **FREE** Personal Network Consultation with Dr. Farallon

At absolutely no cost Dr. Farallon's network experts can help you...

- ✓ Implement switching to optimize your current network
- ✓ Upgrade your Macs, PCs, servers & printers to Ethernet or Fast Ethernet
- ✓ Integrate 10Base-T & 100Base-T with 10/100 hubs & switches
- ✓ Budget for network improvements & plan for network growth

Call: 1-800-613-4954

Visit: www.farallon.com/dr.farallon/mactech.html

"For free, a Dr. Farallon Network Specialist analyzed our existing system, identified a range of options, and helped us plan and install a network with the speed and capacity we need at a price we could afford."

Irene Ogus, CEO,
Media Corps



LIST OF ADVERTISERS

AAA+	25
Aladdin Knowledge Systems Ltd.	5
BeeHive	59
Bowers Development	45
Conix	31
Developer Central	9
Developer Depot	28-29
e-Mediaweekly	IBC
FairCom	65
Farallon	71
IDG Expos-Macworld SF 99	69
Mathemæsthetics, Inc.	1
Metrowerks	BC
NeoLogic	61
NetProfessional	68
Onyx Technology, Inc.	47
OpenBase International	13
PandaWave	23
Quadrivio Corporation	53
Rainbow Technologies	IFC
REAL Software	67
Scientific Placement	68
Stone Design	57
StoneTablet Publishing	41
Tip Top	17
UNI SOFTWARE PLUS	63
USB Stuff	33
Water's Edge Software	51

LIST OF PRODUCTS

ADB Device • BeeHive	59
AppMaker • Bowers Development	45
BugLink • PandaWave	23
c-tree Plus • FairCom	65
Classified • Scientific Placement, Inc.	68
CodeWarrior™ • Metrowerks	BC
Developer Tools • Developer Depot	28-29
Equipment • USB Stuff	33
General Edit • Quadrivio Corp.	53
Joy • AAA+	25
Licenser Kit • Stone Design	57
Mac Publication • e-Mediaweekly	IBC
MachASP • Aladdin Knowledge Systems Ltd.	5
NetProfessional subscription offer • NetProfessional	68
Network Performance Products • Farallon	71
NeoAccess • NeoLogic	61
Object Everything • Tip Top	17
OpenGL • Conix	31
REALbasic • REAL Software	69
Relational Database Engine • OpenBase International	13
Resorcerer® 2 • Mathemæsthetics, Inc.	1
Sentinel • Rainbow Technologies	IFC
Spotlight™ • Onyx Technology, Inc.	47
StoneTable • StoneTablet Publishing	41
Tools Plus™ • Water's Edge Software	51
Trade Show • Developer Central	9
Trade Show • IDG Expos Macworld SF 99	69
VOODOO • UNI SOFTWARE PLUS	63

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

New
Section!



got work?

introducing

E M E D I A W E E K L Y
Careers

In today's exploding digital publishing market, experienced professionals are hard to find. In fact, they're downright scarce. So whether you've got a job to fill or are looking for a job, look no further.

Introducing **eMediaweekly Careers**. A new editorial section in *eMediaweekly* devoted to current hiring trends, training issues, salary levels, recruitment, and the do's and don'ts of hiring. *eMediaweekly* is the **ONLY** magazine to bring these issues to the forefront in the fast-growing digital publishing industry.

Advertising Opportunities

eMediaweekly Careers provides a unique advertising opportunity for companies in search of skilled professionals in content creation. Whether you're

searching for that talented graphic designer or that anal-retentive production director, *eMediaweekly Careers* is the perfect place for you. Only in *eMediaweekly Careers* can you place your ad amidst relevant editorial for the best positioning. And only *eMediaweekly* delivers a core group of 75,000 digital media professionals.

For further information on how you can make use of this invaluable resource, **contact Jennifer Bonwell at 415.243.3680 (East Coast) or Eric Moore at 415.278.8553 (West Coast).**

CAN ONLY BE FOUND IN

«**eMediaweekly**»
THE NEWSWEEKLY FOR DIGITAL MEDIA MANAGERS

Memorandum

TO: Steve Jobs, Interim CEO, Apple Computer, Inc.
Avie Tevanian, Executive Vice President,
Software Engineering, Apple Computer, Inc.

FROM: Greg Galanos, President and CTO, Metrowerks Corp.
Jean Bélanger, Chairman and CEO, Metrowerks Corp.

RE: Mac OS X




Thank you for listening to the Mac community and allowing Mac OS X to preserve Mac developers' codebase. Metrowerks, maker of CodeWarrior, is thrilled to see the momentum back in the developer market and is proud to announce its support and endorsement of Apple's operating system strategy and the next generation of PowerPC architecture. As we did for the Power Macintosh transition, Metrowerks will provide the tools that Mac developers need to make moving to Mac OS X relatively effortless.

And CodeWarrior will continue to move forward, insuring Mac developers have the tools they need to take advantage of Apple's future software products. Since our start in 1994 and fifteen updates later, CodeWarrior Professional Release 3 is our greatest release yet. The CodeWarrior debugger is now fully integrated into the development environment, allowing files to be edited while debugging; we call it Debuggeditor (OK...we're still working on the name), but you can call it enhanced productivity. We've also increased the speed of project loading — loading complex projects is up to five times faster — giving you the efficiency you require.

So again, thank you. We look forward to the future of Mac development and the partnership between Apple Computer, Inc. and Metrowerks Corporation! Coming in 1999, CodeWarrior development tools supporting Mac OS X, AltiVec technology and RAD tools will give Macintosh developers a new level of performance never before seen.

Sincerely,


Greg Galanos
President and CTO


Jean Bélanger
Chairman and CEO

9801 Metrio Boulevard
512.872.4700

Austin

Texas

www.metrowerks.com

78758



CodeWarrior

800-377-5416
www.metrowerks.com